REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NRL Report 7831	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: MICROPROGRAMMED BENCHMARKS FOR THE SIGNAL PROCESSING ARITHMETIC UNIT OF THE AN/UYK-17(XB-1) (V) SIGNAL PROCESSING ELEMENT	5. TYPE OF REPORT & PERIOD COVERED: Interim
	6. PERFORMING ORG. REPORT NUMBER: 5490-245:EF:dlw
7. AUTHOR(s): Harold H. Smith and Leonard E. Russo	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Research Laboratory, Washington, D.C. 20375	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NRL Prob 54B02-06/54B02-10; NASC WF21-241-601; NAVELECSYSCOM XF21-241-019
11. CONTROLLING OFFICE NAME AND ADDRESS: Department of the Navy, Naval Electronics Systems Command, Washington, D.C. 20360	12. REPORT DATE: January 29, 1975
	13. NUMBER OF PAGES: 75
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

All Applications Digital Computer (AADC), AN/UYK-17(XB-1) (V) Signal Processing Element, Microprogrammed Control Unit (MCU), Microprogramming, Signal processing, Signal Processing Arithmetic Unit (SPAU), Signal Processing Element (SPE)

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The AN/UYK-17(XB-1) (V) Signal Processing Element utilizes a Signal Processing Arithmetic Unit (SPAU) to execute subroutines that involve extensive arithmetic operations. Five benchmark programs were written for an early model of the SPAU and run on a simulator. As a result of coding and executing these programs, several changes were made to the SPAU design.

CONTENTS

MICROPROGRAMMED BENCHMARKS FOR THE SIGNAL PROCESSING ARITHMETIC UNIT OF THE AN/UYK-17(XB-1)(V) SIGNAL PROCESSING ELEMENT

INTRODUCTION

During the course of developing the AN/UYK-17(XB-1)(V) Signal Processing Element (SPE) [1], 14 benchmark programs were written (see Appendix A) and executed on Fortran simulators running on the KRONOS Timesharing System. In four of these programs the Microprogrammed Control Unit (MCU) operated independently of the Signal Processing Arithmetic Unit (SPAU); in five programs the MCU performed data transfer and control operations while the SPAU did basic signal processing operations; and in the other five programs the MCU did data transfers and called the SPAU to do combinations of signal processing operations. This document describes the five basic signal processing operations used in these benchmarks as they were programmed in the SPAU. The MCU benchmark programs are described elsewhere [2].

The AN/UYK-17 SPE is a digital processor that has been value engineered for the execution of signal processing algorithms required in military weapons and sensor systems. The SPE is microprogrammable so that it can be tailored to perform processing for particular system applications. The SPE can be used alone but is intended to be implemented as a component of the All Applications Digital Computer (AADC) currently under development by the Naval Air Systems Command.

OBJECTIVES

A central theme in the SPE program has been the idea of developing software architectural models prior to commission of the SPE design to hardware. In this way design improvements could be developed during programming and easily fed back into the architecture design. In the process of coding the benchmarks, certain desirable features were added to the design. These are described in the last section of this report. The benchmarks provided measures of SPAU performance in terms of clock cycles required for execution and number of instructions required. Finally, the programs demonstrated the coding of the SPAU to operate as a pipelined processor.

Manuscript submitted October 3, 1974.

DETAILED DESCRIPTIONS

The benchmark programs described herein (see also Appendix B) perform the following functions:

1. A fast Fourier transform (FFT) [3] of an array of 4096 complex values, whose 16-bit real and imaginary parts are in adjacent halves of each buffer memory word.

2. An FFT of an array of 1024 real values that occupy consecutive 16-bit halves of 512 buffer memory words.

3. A coherent demodulation and octave filtering [4] of an array of 1024 8-bit real points packed four to a buffer memory word. At each octave, the input array is "decimated" by a factor of 2 and split into quadrature channels by multiplication by sine and cosine time functions. Four-pole low-pass filters (LPF) are used in all 16 output channels, and 7 filters are used between octaves.

4. A complex weighting (sum of complex products) of groups of four consecutive complex points, selected from an input array of 256 points. The array of 1024 weights is irregularly addressed with respect to the corresponding input points.

5. A cross correlation [5] of points from 32 channels with 32 beams and steering delays over 1024 frequencies. Each output point is the product of three complex factors. The input channel and beam data have 32-bit precision; the output data have 16-bit precision.

Complex FFT

The program LFFT performs a fast Fourier transform on an array of N complex points where N is a power of 2 and lies between 16 and 4096, inclusive. It is used in the benchmark programs to transform 4096 complex points and to transform 1024 real points as a packed 512 point array (see the following subsection). The output values, denoted by $A(n)$, are complex (16 bits real and 16 bits imaginary) and are derived from the complex input points, denoted by $X(j)$, according to the discrete Fourier transform (DFT) equation

$$A(n) = \sum_{j=0}^{N-1} \times (j) \exp\left(\frac{2 \pi i j n}{N}\right), \tag{1}$$

wherein $i = \sqrt{-1}$ and N is the number of input points (n = 0, 1, ..., N-1).

This equation is evaluated by means of a "variable shuffle" algorithm that is derived in Appendix C [6,7]. The DFT may be expressed in matrix form as

$$\mathbf{A} = \mathbf{W}_N \mathbf{X} \tag{2}$$

where **A** and **X** are column vectors:

$$\mathbf{A} = [A(0)\ \ A(1), \ldots A(N\text{-}1)]^T \tag{3}$$

$$\mathbf{X} = [X(0)\ \ X(1), \ldots X(N\text{-}1)]^T. \tag{4}$$

The superscript T denotes a transpose of rows and columns. The element of matrix $\mathbf{W}_N$ at the nth row and jth column is

$$W_N\,(n,j) = \left[\exp\left(\frac{2\,\pi\,i}{N}\right) n\,j \text{ modulo } N\right]. \tag{5}$$

The algorithm is based on a radix-2 factorization of $\mathbf{W}_N$, but unlike many FFT algorithms it does not require bit-reversed reordering of the input or output array. It does, however, require a data work area in memory that is twice the size of the input array because the shuffling cannot be done "in place." The procedure consists of three operations, each such triplet constituting a single stage:

1. Combine the input terms (to a stage) by forming sums and differences of pairs.

2. Shuffle the terms in a manner similar to the shuffling of playing cards. In the first stage the array is "cut," and single terms are interleaved; in the second stage pairs of terms are merged.

3. Multiply the terms by trigonometric weights.

The operations appear as separate factors in the expansion of $\mathbf{W}_N$:

$$\mathbf{W}_N = \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{K}_{N/2} \end{bmatrix} \mathbf{S}_{N/4} \quad \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & & \\ & \ddots & \\ & & \mathbf{K}_{N/4} \end{bmatrix} \mathbf{S}_{N/8} \ldots \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & & \\ & \ddots & \\ & & \mathbf{K}_2 \end{bmatrix} \mathbf{S}_1 \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} . \tag{6}$$

The **S** matrices perform the shuffle operations, **I** denotes the identify matrix (of various dimensions, see Appendix C), and the **K** represents the weighting factors as diagonal matrices:

$$\mathbf{K}_{N/2} = \text{diag}\ (w^0\ w^1 \ldots w^{N/2-1}) \tag{7}$$

$$\mathbf{K}_{N/4} = \text{diag}\ (w^0\ w^2 \ldots w^{2(N/4-1)}) \tag{8}$$

where $w = \exp\ (2\ \pi\ i/N)$.

Each complex sum and difference operation on a pair of terms may be represented in a butterfly diagram, as depicted in Fig. 1. One stage of the FFT consists of $N/2$ such butterflies, and the entire FFT involves $\log_2 N$ stages. In the first stage the input points to a butterfly, P_1 and P_2, are obtained from the input data vector **X**; and in the final stage the output points, $P_1{}^1$ and $P_2{}^1$, are elements in the output vector **A**. The first three variable shuffle operations are depicted in Fig. 2.

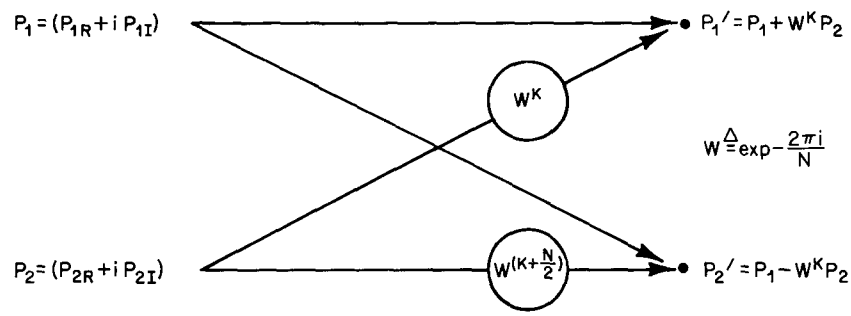


Fig. 1 — FFT Butterfly diagram

Data Structure

The principal data structures employed in this macro are the input and output data areas, both of which are in BUFA; the working data area of BUFB; the linkage data in *W* addresses 0 through 4; and working data locations in *W* store. The data are treated as complex 32-bit words in buffer, with the real parts in the 16 most significant bits and the imaginary parts in the 16 least significant bits. On input to Stage 1 the data are read from BUFA starting at the Data Address, which is contained in the linkage information provided by the MCU. Output data from Stage 1 are stored in BUFB starting at the Result Address, which is also part of the linkage from the MCU. On succeeding stages data are both read and written from/to buffer areas that begin at the Result Address.

W Store —

Address No.	Content
0	Control store starting address of LFFT
1	One-half the number N of input points
2	$\log_2 N$
3	Data Address
4	Result Address (end of linkage)
5	Number of butterflies in a half group
6	Number of groups per stage
7	Unused
8	Read only memory (ROM) address increment
9	One

Coefficient Store — Addresses 0 through 1024 contain the sine and cosine of angles between 0° and 90°, inclusive, at equal increments in angle. The sine is contained in the 16 most significant bits and the cosine in the 16 least significant bits.

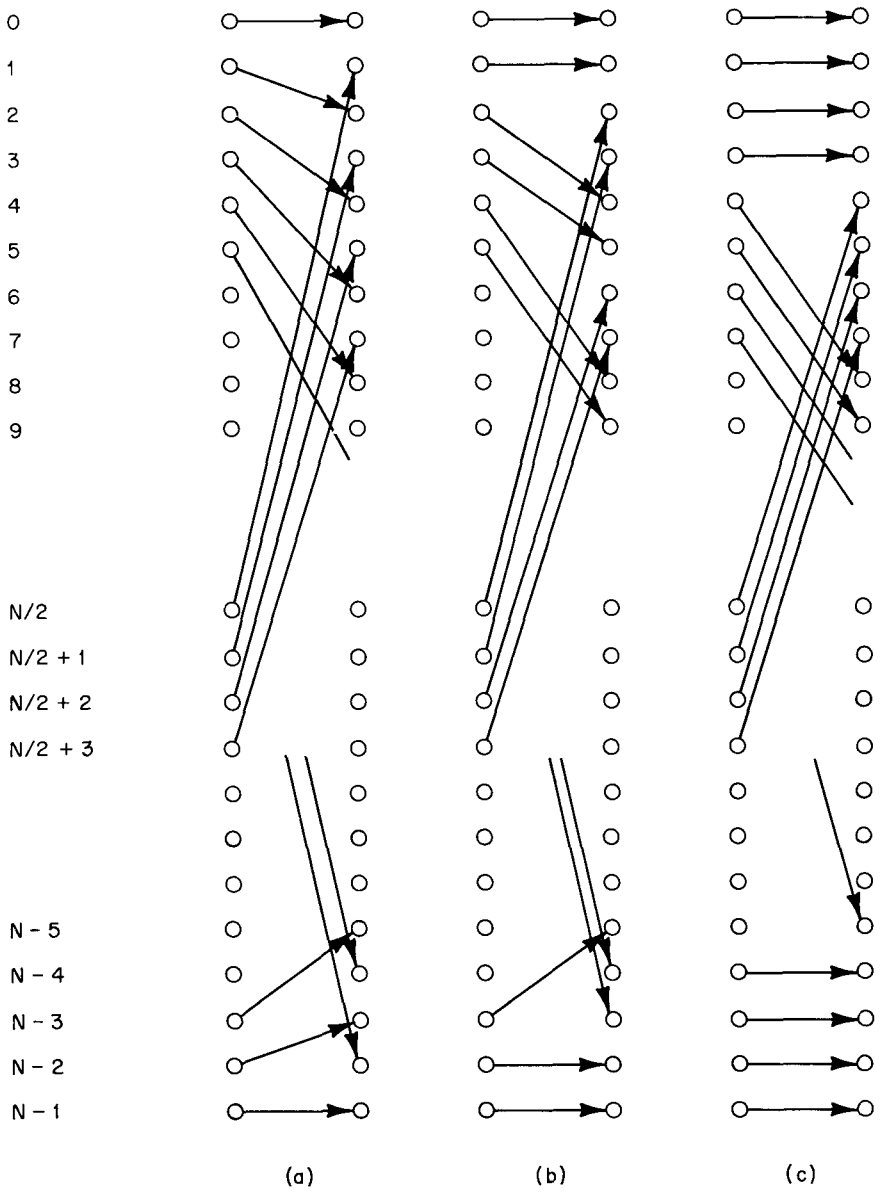


Fig. 2 — Interstage shuffle: (a) First operation, (b) Second operation, and (c) Third operation

Program Description

A Flow graph of a single butterfly calculation, as carried out in the SPAU, is shown in Fig. 3. The sequencing of successive stages is shown in the flowchart of Fig. 4. Stage 1 involves angular exponents of 0° for all j, hence the values are read from ROM just one time. Stage 2 uses the 0° and 90° values and therefore alternates between ROM address 0 and ROM address 1024. Each successive pair of butterflies uses the same angles. In Stage 3 the same angles are used by groups of 4 butterflies, and in later stages by groups, of 8, 16, and so forth. In Stage 3, the effect of angles in the second quadrant (i.e., 135°) is taken into account by changing additions to subtractions or subtractions to additions. Finally, in the General Stage, there are two distinct loops within the stage: for acute angles the ROM address is constantly incremented until it reaches 1024, and for obtuse angles it is decremented and some of the operations are changed in sign to account for the second quadrant.

Input data are always read from the buffer on channel A (BUFA) and output data are stored in the buffer on channel B (BUFB). At the end of each stage the roles of the buffers are reversed by giving a SWAP command. The starting address of the initial problem data in BUFA is arbitrary, as is the starting address of the output results, which ultimately are also in BUFA. Because two data areas are available, an out-of-place algorithm is used to implement the FFT, and bit-reversed reordering of the input or output points is avoided. The algorithm requires that the sequencing of the points be shuffled between stages, which is done "on the fly" by properly generating "write" addresses.

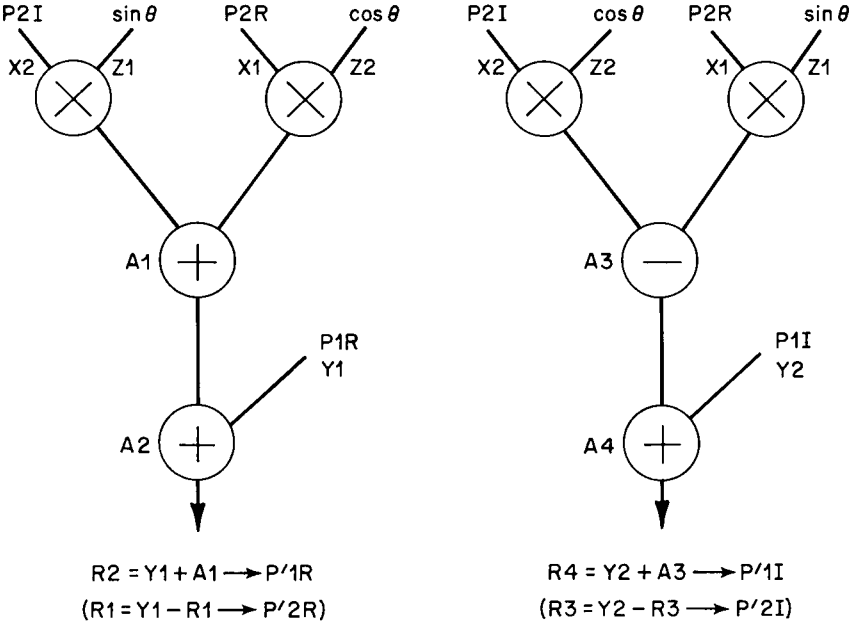


Fig. 3 — Butterfly flow graph

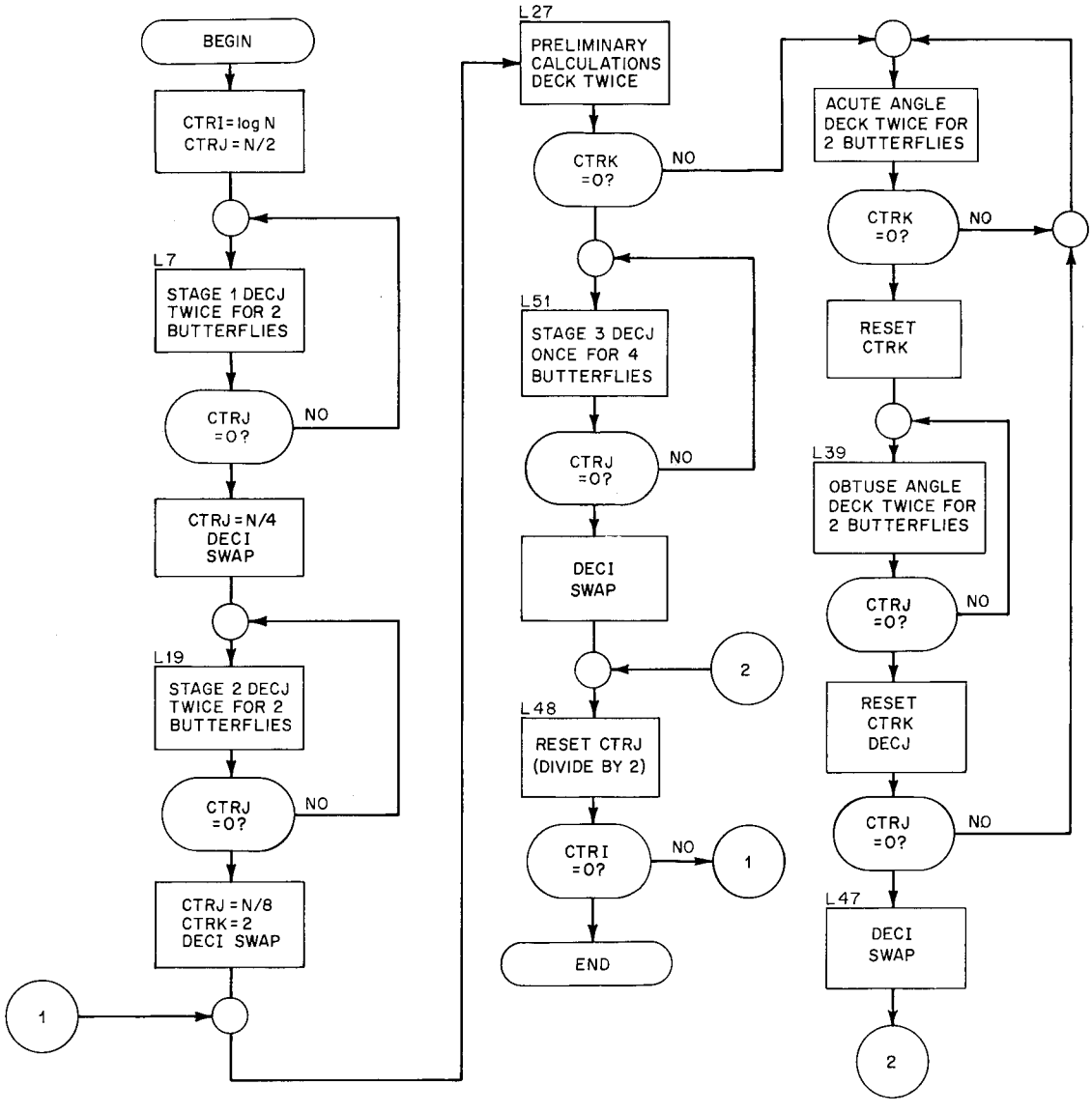


Fig. 4 — Flowchart of LFFT

At the end of each Stage 1 operation, the output points from a butterfly are written into interleaved addresses in BUFB, in the manner shown in Fig. 2(a). The effect is as though the original array in BUFA were "cut" in two, as a pack of playing cards, and shuffled on a one-for-one basis, each point from the first half of the array being followed by a corresponding point from the second half of the array. In Stage 2 the points are shuffled in pairs, as shown in Fig. 2(b), in Stage 3 in quadruplets, as shown in Fig. 2(c), and in groups of increasing powers of 2 in succeeding stages. At the end of the final stage no additional shuffle is necessary: the points are in correct output sequence.

The first three stages of the FFT are programmed as distinct loops, and all succeeding stages are programmed as a fourth type of loop. The latter consists of two subloops, one for acute angles and one for obtuse angles. The first three stages are each terminated under the control of CTRJ by counting the number of groups in the stage. The number of butterflies within a group (respectively one, two, and four) is implicit in the coding. In the fourth and succeeding stages, the number of butterflies is first counted to CTRK to determine the end of a group, and then the number of groups is counted by CTRJ to determine when the stage ends. The end of the FFT is indicated by CTRI, which counts down from $\log_2 N$.

In the obtuse angle subloop, beginning at L38, the signs of the operations involving P2, P4, and A3 are reversed to effectively change the sign of the cosine in the second quadrant. In all stages the components of the butterfly output points are generated in registers R1 through R4 and are packed before outputting, first from register pair R2R4 as a single 32-bit word, and then from R1R3. The two input points to any butterfly are spaced $N/2$ positions apart, controlled by INCA. First, the lower point P2 is read, then INCA is subtracted from the current contents of BARA, and the upper point P1 is read. The shuffled output is controlled by the sequencing of BARB, which, in the first stage, amounts to a constant increment by one address. In the second stage and thereafter, INCB is set equal to the number of butterflies per group for that stage, which is the output separation between the two points in any butterfly in that stage. At the end of each group, BARB is incremented by one to begin the next group output.

Performance

The program was run to calculate transforms for data arrays involving 32, 64, 128, 256, 512, and 1024 complex points. Table 1 summarizes the number of clock pulses required for each of these runs; the corresponding real time elapsed (at 150 ns per clock interval); and the percent of clocks expended in overhead operations, obtained by subtracting $N \log_2 N$ from the total number of clock pulses and dividing by $N \log_2 N$, where N is the number of data points.

Table 1
Performance Time

Number of Complex Data Points	Total Number of Clock Pulses	Time (μs)	Percent Overhead
32	192	28.8	20
64	423	63.5	10
128	941	141	5
256	2,100	315	2.5
512	4,668	700	1.3
1,024	10,307	1,550	0.6

Real FFT

A discrete Fourier transform of real data may be implemented using the FFT with considerable savings in execution time if the conjugate symmetry of the spectra of real data is exploited. The real FFT (REFFT) is a SPAU macro that exploits those symmetries. The input to REFFT is an $N/2$-point complex array formed from an N-point real array by—

1. Sequentially packing odd-indexed elements of real array into imaginary part of complex array and even-indexed elements of real array to real part of complex array.

2. Fourier transforming $N/2$-point complex array using LFFT. N is any power of 2 in the interval from 16 to 4096.

REFFT unpacks the $N/2$-point array and completes the processing needed to output an N-point spectrum array.

An N-point DFT is defined as follows:

$$F_k = \sum_{m=0}^{N-1} e^{-j2\pi mk/N} X_m ; \quad k = 0, 1, ..., N-1, \tag{9}$$

where $j = \sqrt{-1}$.

It is desired to solve this for the case $\{X_m\}$ real. Equation (9) may be expressed as the sum of even- and odd-indexed terms:

$$F_k = \sum_{m=1}^{N/2-1} X_{2m} e^{-j2\pi k 2m/N} + \sum_{m=0}^{N/2-1} X_{2m+1} e^{-j2\pi k(2m+1)/N} \tag{10}$$

Let $N' = N/2$; $k = pN' + \ell$; $p = 0, 1$; $\ell = 0, 1, \ldots, N'-1$; then

$$F_{pN'+\ell} = \sum_{m=0}^{N'-1} X_{2m} e^{-j2\pi(pN'+\ell)2m/N} + \sum_{m=0}^{N'-1} X_{2m+1} e^{-j2\pi(pN'+\ell)(2m+1)/N}$$

$$= \sum_{m=0}^{N'-1} X_{2m} e^{-j2\pi m\ell/N'} + (-1)^p e^{-j2\pi\ell/N'} \sum_{m=0}^{N'-1} X_{2m+1} e^{-j2\pi\ell m/N'}$$

$$= F'_\ell + (-1)^p e^{-j2\pi\ell/N'} F''_\ell. \tag{11}$$

Here, F'_ℓ is the $\ell + N$ frequency component of the $N/2$-point transform of $\{X_{2m}\}$; F''_ℓ is the ℓth frequency component of the $N/2$-point transform of $\{X_{2m+1}\}$. Knowing F'_ℓ and F''_ℓ for all ℓ, it is possible to find $\{F_k\}$, the frequency array. Equation (11) is the last stage of butterflies for the N-point FFT.

Up to now, the fact that X_m is real has not been used. $\{X_m\}$ real provides a neat way of computing $\{F'_\ell\}$ and $\{F''_\ell\}$ with one $N/2$-point complex FFT. To do this, the odd components of $\{X_m\}$ are packed in ascending order into the imaginary part of a complex $N/2$ point array $\{Z_m\}$. The even components of $\{X_m\}$ are packed in ascending order into the real part of $\{Z_m\}$.

Mathematically,

$$Z_m = X_{2m} + jX_{2m+1} \quad m = 0, \ldots, N' - 1. \tag{12}$$

Transforming both sides of Eq. (12) yields

$$\begin{aligned} f_{N/2}\{Z_m\} &= f_{N/2}\{X_{2m}\} + jf_{N/2}\{X_{2m+1}\} \\ &= \{F'_\ell\} + j\{F''_\ell\} \end{aligned} \tag{13}$$

where $F_{N/2}$ is the DFT of the $N/2$ point array in brackets. From Eq. (13) it is seen that

$$T_\ell = F'_\ell + jF''_\ell \tag{14}$$

where T_ℓ is the ℓth term of $f_{N/2}\{Z_m\}$.

Because $\{F_\ell'\}$ and $\{F_\ell''\}$ are transforms of real sequences, it follows that

$$F_\ell' = F'^*_{N/2-\ell} \tag{15}$$

$$F_{\ell}'' = F'''^{*}_{N/2-\ell} \ . \tag{16}$$

The transforms $\{F_{\ell}'\}$ and $\{F_{\ell}''\}$ are complex but may be separated by using Eqs. (15) and (16). Let T_{ℓ} be the ℓth frequency component in $F_{N/2}\{Z_m\}$; then

$$F_{\ell}' = \frac{T_{\ell} + T^{*}_{N/2-\ell}}{2} \tag{17}$$

$$F_{\ell}'' = \frac{T_{\ell} - T^{*}_{N/2-\ell}}{2i} \ . \tag{18}$$

Thus, REFFT will receive an $N/2$-point complex array that has been packed and then transformed using LFFT. REFFT isolates arrays $\{F_{\ell}'\}$ and $\{F_{\ell}''\}$ as in Eqs., (17) and (18), then completes the necessary butterfly operations in Eq. (11) to yield an N-point array, the transform of the original real sequence $\{X_m\}$.

Data Structure

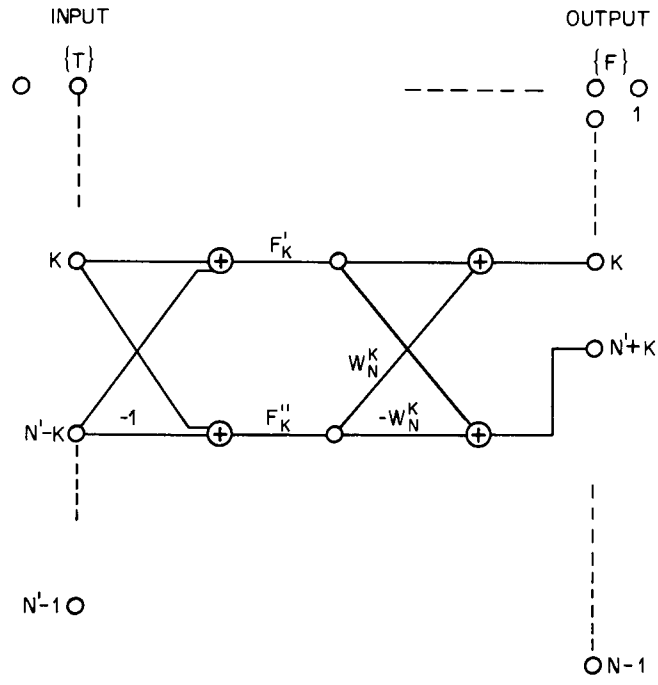
REFFT was written to be consistent with LFFT in the location of W store parameters. The parameters left in W store by a call to LFFT for an $N/2$-point array are just those needed by REFFT. The W store parameters are —

W(3): address of data array in BUFA
W(4): address of work space in BUFB
W(1): $N/4$, consistent with LFFT for $N/2$ points
W(2): $m-1$, consistent with LFFT for $N/2$ points
W(8): $4096/N$, consistent with LFFT for $N/2$ points.

The N-point complex transform array will be located in BUFB starting at the location specified in W(3). The input data of $N/2$ complex points reside in BUFA starting at location in W(3); they are processed to yield an unscaled transform array in BUFB work space beginning at location in W(4). The transform array is scaled to give a scaling identical to that of LFFT operating on the same real input array. Finally, the scaled N-point transform array is located in BUFA starting at location in W(3).

Program Description

The input to REFFT is a complex array of $N/2$ points; it is the DFT of the complex $N/2$ point array formed by placing even points of $\{X_m\}$ in the real part and odd points of $\{X_m\}$ in the imaginary part where $\{X_m\}$ is the real array that is to be transformed. There are two main loops needed in REFFT to provide the necessary range of phase angles in Eq. (11) with only a 90° read-only memory. In each loop, F'_{ℓ} and F_{ℓ}'' are isolated, then a butterfly is performed on the isolated values. Finally, the data array is scaled by 1/2, a procedure that takes $N-4$ clocks. See Figs. 5 and 6.



$W_N^K = \exp(-j2\pi K/N)$

T_0 and $T_{N/2}$ are treated as special cases. For example,

$$T_0 = \sum_{m=0}^{N'-1} X_{2m} + j\sum_{m=0}^{N'-1} X_{2m+1}; \text{ thus,}$$

$$F'_0 = \sum_{m=0}^{N'-1} X_{2m}$$

$$F''_0 = \sum_{m=0}^{N'-1} X_{2m+1}$$

may be obtained trivially since $W_{N/2}^0$ is real.

Input $\{T\}$ is an $N/2 = N'$ point complex array. Output $\{F\}$ is an N-point complex array.

Fig. 5 — REFFT addressing and processing

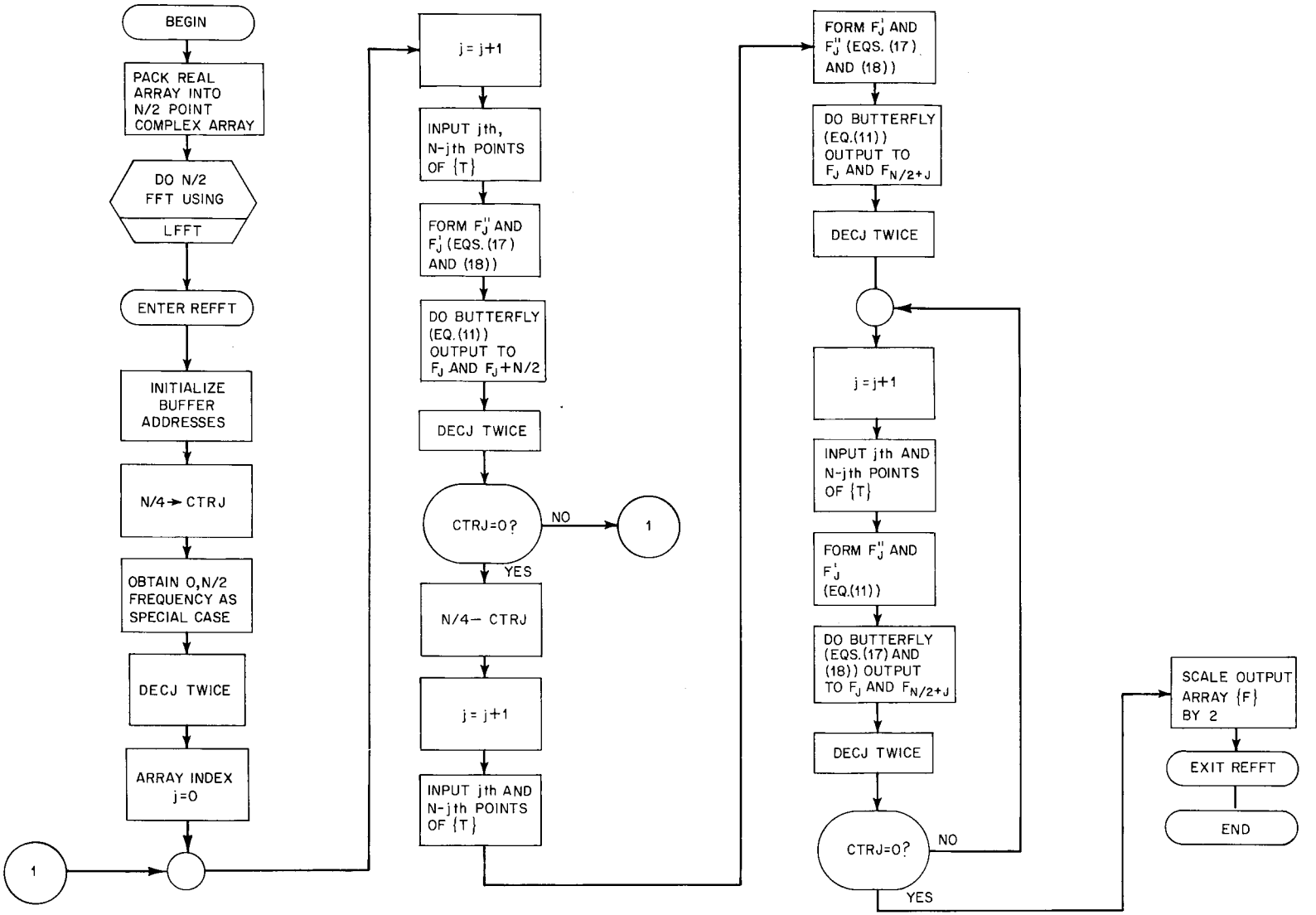


Fig. 6 — Flowchart of REFFT. Two loops are necessary to step up and down 90° ROM giving effective exponent range, 0 to π. We address ROM for exponential factors in Butterfly (Eq. (11)).

Performance

The times given assume packed data, but include the time for the $N/2$-point FFT using LFFT. A real 1024-point FFT was run in 7241 clocks, of which 1020 are for scaling and 4668 are for a 512-point complex FFT. In general, a real N-point transform takes $5N/2+13+$ LFFT$(N/2)$ clocks. N-4 of these clocks are for scaling.

COHERENT DEMODULATION AND OCTAVE FILTERING

The macro OCT performs complex demodulation and octave filtering of a sequence of real data points in the manner shown in Fig. 7. The input sequence is repeatedly filtered and heterodyned to form eight octaves of complex data points. All of the filters are recursive and consist of pairs of cascaded two-pole, two-zero modules. A block diagram of the basic module is shown in Fig. 8. The output sequence of every filter is "decimated": three out of every four output values are deleted in the case of the quadrature channel filters and one out of every two output values is deleted for the filters between octaves. In the benchmark program, 1024 real data samples are input initially so that the output in octave 8 consists of 256 complex points, in octave 7 128 points, and so forth down to octave 1, which contains only 2 complex points.

Filter Implementation

As is shown in the filter block diagram of Fig. 8, the delayed terms W_1 and W_2 are related to the initial feedback term W_0 by the delay operator z^{-1} [4]:

$$W_1 = z^{-1}W_0$$

$$W_2 = z^{-1}W_1 = z^{-2}W_0.$$

The filter output is

$$\text{OUT} = W_0 + A_1W_1 + A_2W_2 = W_0(1+A_1z^{-1} + A_2z^{-2}).$$

In terms of W_0, the input to the filter is

$$\text{IN} = W_0 - B_1W_1 - B_2W_2 = W_0(1 - B_1z^{-1} - B_2z^{-2}).$$

The filter implementation used in OCT differs slightly from Fig. 8 in that the feed-forward term from W_0 has been moved to the input side of the adder; it connects to IN rather than W_0. This is accomplished by substituting for W_0 in the expression for OUT. The two quantities generated by the two-pole module are

$$\text{OUT} = \text{IN} + (A_1+B_1)W_1 + (A_2+B_2)W_2$$

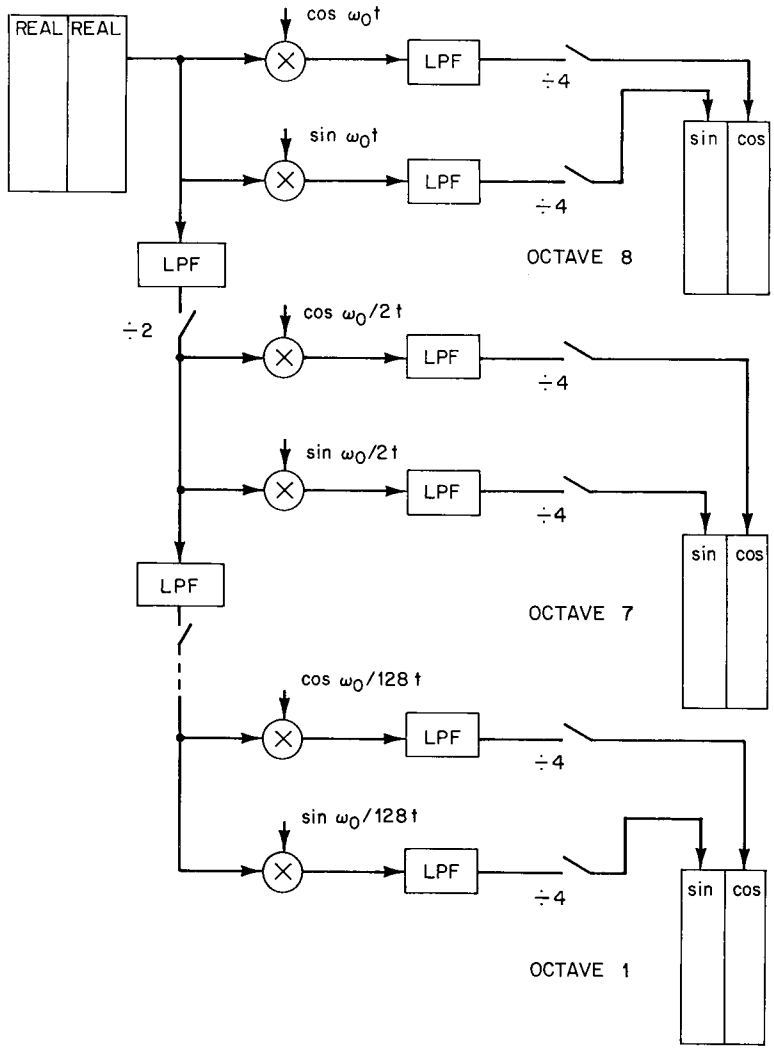


Fig. 7 — Octave Filter block diagram

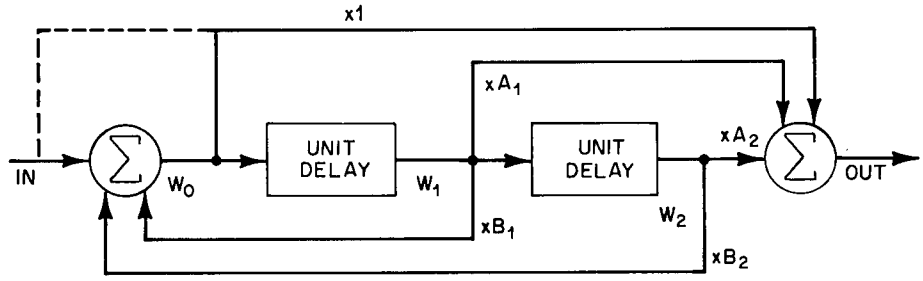


Fig. 8 — Two-pole block diagram for LPF

$$W_0 = \text{IN} + B_1 W_1 + B_2 W_2.$$

The flow graph for the implementation of these equations is shown in Fig. 9. The four coefficients, B_1, B_2, A_1+B_1, and A_2+B_2, are stored in ROM and applied to multipliers 1 to 4, respectively. The input value IN is read from buffer A into the Y store and is applied to adders 2 and 4. The output of adder 2 is the W_0 term, which is delayed by storing it in the X store. Successive values of W_0 are alternated between X_1 and X_2; in each instance W_0 replaces what had been W_2 and the previous W_1 becomes the new W_2. The output of adder 4 is the output quantity from the two-pole module. In the case of the first two pole module of a pair, it is used immediately as the Y input to the second two pole.

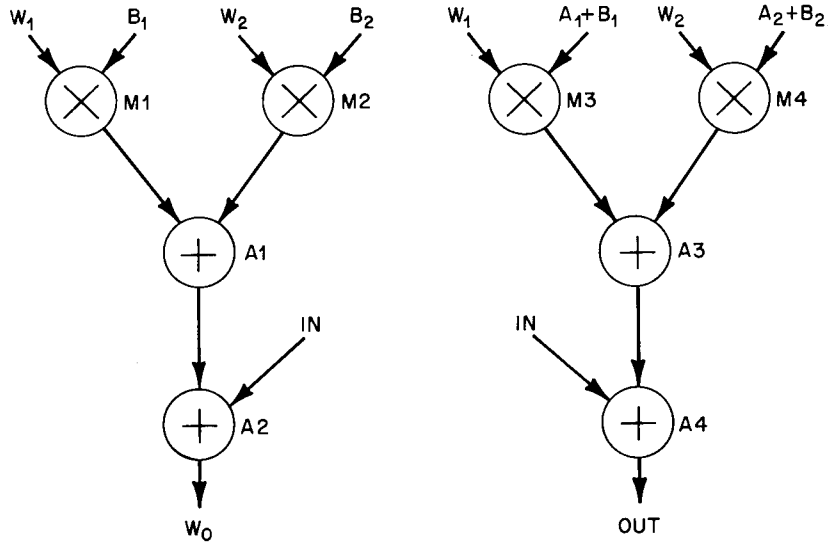


Fig. 9 — Two-pole flow graph

In choosing a value for f_0, the frequency of the heterodyne oscillator in octave 8, the following relationships have been assumed between f_0 and W, the bandwidth of the input sequence, and f_s, the sampling frequency:

$$f_0 = \frac{3W}{4}$$

$$f_s = \frac{5W}{2}.$$

This corresponds to a 25% safety margin in the Nyquist rate (5/4 times $2W$). The ratio of f_s to f_0 is thus 10/3, so that after every 10 samples, the oscillator has completed three cycles and returned to its starting phase. The effect of heterodyning is achieved by storing 10 sine/cosine pairs of values in ROM for the 10 different phase angles corresponding to every 10 sample points and using these repeatedly as multiplying factors to be paired with the input data stream. The angular increment between samples is $6\pi/10$. In each succeeding octave the oscillator frequency is reduced by a factor of 2, but the interval between input samples is increased by a factor of 2, so that the phase angles are the same in every octave, and the same 10 sine/cosine pairs are used in the multiplications in all octaves.

Data Structures

The principal data structures employed in this macro are the input and output data areas, both in BUFA (the buffer memory connected to channel A); the linkage data, transmitted from the MCU into the W store; and the working areas, which consist of parts of BUFA, BUFB, and the W store. The following subsections describe the assignments for the fixed number of data points (1024) required for the benchmark test.

Buffer Memory A — The 1024 real data values, packed two to a word, with the earlier point in the left (more significant) half.

Address No.	Content
512 - 767	Octave 8 output
768 - 895	Octave 7 output
896 - 959	Octave 6 output
960 - 991	Octave 5 output
992 - 1007	Octave 4 output
1008 - 1015	Octave 3 output
1016 - 1019	Octave 2 output
1020 - 1021	Octave 1 output

During the execution of the macro, addresses 0 to 255 are written over as temporary storage for the outputs of the real-channel filters as follows:

Address No.	Content
0 - 255	Decimated output of octave 8/7 real filter
0 - 127	Decimated output of octave 7/6 real filter
0 - 63	Decimated output of octave 6/5 real filter
0 - 31	Decimated output of octave 5/4 real filter
0 - 15	Decimated output of octave 4/3 real filter
0 - 7	Decimated output of octave 3/2 real filter
0 - 3	Decimated output of octave 2/1 real filter

Buffer Memory B —

Address No.	Content
0 - 1023	Demodulator output, octave 8
0 - 511	Demodulator output, octave 7
0 - 255	Demodulator output, octave 6
0 - 127	Demodulator output, octave 5
0 - 63	Demodulator output, octave 4
0 - 31	Demodulator output, octave 3
0 - 15	Demodulator output, octave 2
0 - 7	Demodulator output, octave 1

W Store —

Address No.	Content
0	Control store starting address of OCT
1	Number of input points (end of linkage)
3	Second ROM address of most recently used filter
4	Zero
5	One
6	Latest output address of complex filter
7	Latest input address of real filter
8	Latest output address of real filter
9	Five, initialization value for oscillator phase counter
10	Initialization value for input word counter (offset)

Coefficient Store —

Address No.	Content
2049-2050	Complex filter coefficients, octave 8
2051-2052	Real filter coefficients, octave 7
2053-2054	Complex filter coefficients, octave 7
2055-2056	Real filter coefficients, octave 6
2057-2058	Complex filter coefficients, octave 6
2059-2060	Real filter coefficients, octave 5
2061-2062	Complex filter coefficients, octave 5
2063-2064	Real filter coefficients, octave 4
2065-2066	Complex filter coefficients, octave 4
2067-2068	Real filter coefficients, octave 3
2069-2070	Complex filter coefficients, octave 3
2071-2072	Real filter coefficients, octave 2
2073-2074	Complex filter coefficients, octave 2
2075-2076	Real filter coefficients, octave 1
2077-2078	Complex filter coefficients, octave 1
2079	Sine/cosine of 0, $3\pi/5$ rad
2080	Sine/cosine of $6\pi/5$, $9\pi/5$ rad
2081	Sine/cosine of $12\pi/5$, $15\pi/5$ rad
2082	Sine/cosine of $18\pi/5$, $21\pi/5$ rad
2083	Sine/cosine of $24\pi/5$, $27\pi/5$ rad

Program Description

An understanding of the operation of the program is aided by making reference to the flowchart of Fig. 10. The left column lists the major elements that may be active in a SPAU instruction. The right column indicates the content of the instructions that comprise the program. The first three instructions set the INC registers to one and set up $W(3)$, $W(5)$, and $W(6)$. The main program is a four-instruction loop that generates eight subroutine calls and is followed by an interrupt return of control to the MCU. The remainder of the program contains coding for performing the filtering of real sequences (RFIL), complex sequences (CFIL), and complex demodulation (DEMOD). Octave 8 does not require real filtering, so the program enters at DEMOD, where both buffer addresses are set to zero and the ROM address is set to that of the first of the sine/cosine coefficients. Consecutive input points are read into $X(0)$ and $Y(0)$ and multiplied by the 10 successive coefficients. The resulting quadrature sequences are packed into consecutive words in BUFA, starting at address zero, with the sine component in the left (more significant) half, and the cosine component on the right. Counter K is decremented from five to zero as every five input words (i.e., 10 data points) are processed. Counter I is set to the number of input words less three (to accommodate three read operations that occur before the main loop begins). The main loop of DEMOD occupies addresses 27 through 30. Each pass through the loop

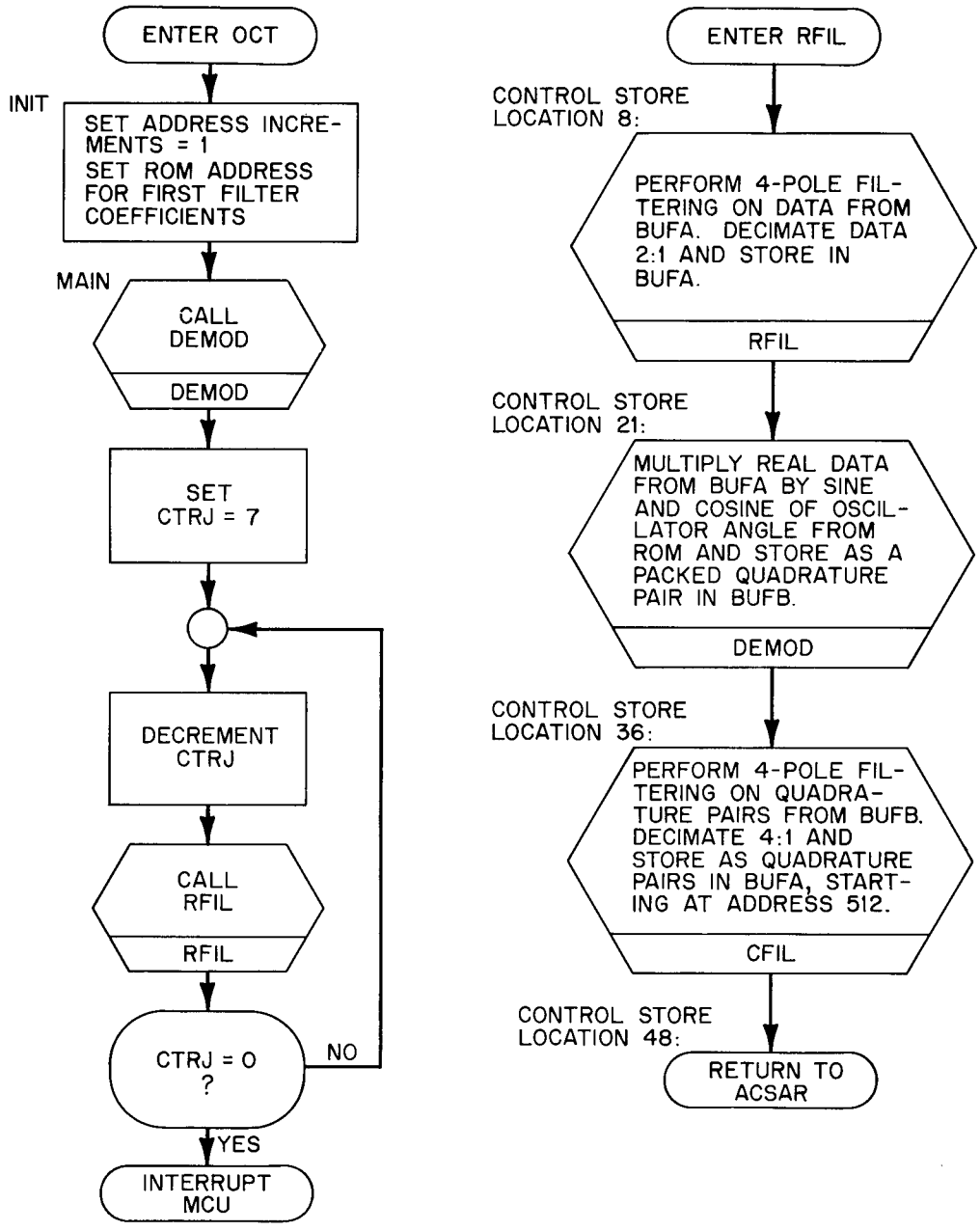


Fig. 10 — Octave filter flowchart

generates a quadrature pair (one 32-bit word in BUFB) for each of four packed data words that are read in from two consecutive locations in BUFA. The number of points input should therefore be a multiple of 4.

The beginning of the complex filter CFIL overlaps the end of DEMOD to address the next set of filter coefficients. Sine-channel feedback terms are held in store X at addresses 1 and 2; cosine-channel terms are at addresses 3 and 4. The roles of X_1 and X_2 are reversed in the multiplications in control store addresses 42 through 45. After a two-clock delay, the multiplier products are available to form the W_0 and OUT terms of Fig. 9. This is done in the adders, concurrently with the multiplications, starting at address 40. One complex point is output to BUFA for every four complex points read in from BUFB. Indexing of the input stream is maintained by CTRI, which is initialized from $W(1)$ without changing $W(1)$, because it will be needed unchanged to index RFIL. CTRK is used to control the "decimation" from four input points to one output point. The OUT term for the first member of each pair of input points is not generated, and for the second number it is selected only on alternate pairs.

Coding for the real filter RFIL is in control store locations 8 through 20, where it acts as an input filter for successive octaves. The input data are read from BUFA starting at address 0, and the feedback terms in locations 1 and 2 of the X store are initially set to 0. The multiplications for the first two-pole module of the filter occur at locations 11, 14, and either 17 or 18. At 17 a real output point is packed into the left half of a buffer word using register R3. The following input point is not processed to give an output, only feedback terms, at location 14. The next input point is packed into the right half of a buffer word at 18, using register R4. This accomplishes decimation by a factor of 2, under control of CTRK. The MAIN program loop calls RFIL seven times, under control of CTRJ, and then returns an interrupt to the MCU. Scaling of the output data for display purposes is done during the MCU data transfers.

Performance

The program was run using input arrays of 256, 512, and 1024 real points, and the cumulative number of clock pulses required to process each octave for these arrays is given in Table 2. A single LPF design (eight coefficients) was used for all the filters; it has a cutoff frequency at one-half the sampling frequency.

Table 2
Clock Pulses Required for Processing

Octave	Number of Input Points		
	256	512	1024
1	1,293	2,573	5,133
7	2,717	5,405	10,781
6	3,437	6,829	13,613
5	3,805	7,549	15,037
4	3,997	7,917	15,727
3	4,101	8,109	16,125
2	—	8,213	16,317
1	—		16,422

The benchmark problem, involving 1024 input points, requires 16,422 clock pulses, equivalent to 2.46 ms.

Complex Weighting

This program, WGTSUM, creates an array of weighted sums of four complex terms. Each term is a complex product: one input to this product comes from a coefficient table; the other input comes from a table that is addressed indirectly using another table. Thus, three tables in buffer are involved, one containing addresses, the other two data. The mathematical expression programmed is

$$Y_i = \sum_{k=1}^{4} C_{i,k}\ X_{Ji+k-1} \quad i = 1, \ldots, 256 \tag{19}$$

where X is the input array, Y is the output array, and $C_{i,k}$ is the $(4i+k)$th entry in the table of complex coefficients C.

Data Structure

The address of C is in $W(4)$ and C resides in BUFB. J is the Indirect Address Table (IAT) located in BUFA. Its location is in $W(3)$. X is the data table. X_{Ji} is the element formed by using the ith entry in J as an address. $W(2)$ contains the number of points in J, assumed to be 256 in this problem. Locations $X(1\text{-}5)$ and $Y(1\text{-}5)$ are used for input buffering and temporary storage. Y_i, the output value, is written over J^i, the ith location in J. Thus, the IAT is destroyed by this program. Both X and J reside in BUFA.

Program Description

Tables J and C are addressed sequentially (see Fig. 11). For each value of i, there are four coefficients $C_{i,k}$. For each J_i, one obtains a pointer to a block of four data points in the X table. For each output Y_i, X_{Ji+k-1} and $C_{i,k}$ are multiplied and summed over k. Entries in X and C are complex.

One output is computed for each pass through a six-clock loop L0 to L5. Six clocks are needed for four complex multiplies and the transfer of the indirect address located in the J_i from BUFA to BARA. $W(5)$ is the local storage that saves the current address of the BUFA data array.

Because of SPAU hardware design, there is no direct path from the buffer to the W store; hence, a clock per loop is used to transfer the indirect address word to the Address Generator.

Performance

A table of 256 elements with a 1024-entry coefficient table took 1543 clocks to output 256 weighted points, about 1 point per 6 clocks.

Crosscorrelation

This program, XCORREL, performs a beam output crosscorrelation. The values of $\varphi_{\ell m}$ are to be evaluated for all ℓ and m values:

$$\varphi_{\ell m}(\omega_k) = Y_m(\omega_k) ej\omega_k T_{\ell m} X_\ell(\omega_k) \tag{20}$$

where

$\ell = 1, ..., 32,$
$m = 1, ..., 32,$
$k = 1, ..., 1024.$

The MCU would set up the SPAU for each steering delay; i.e., for each value of ℓ and m. The SPAU will implement the crosscorrelation; XCORREL is the solution of Eq. (20) for one fixed steering delay (i.e., one value each of ℓ and m). The solution of Eq. (20) will be achieved by using the DFT and relying on the fact that translation in time produces a linear phase shift in the frequency domain.

A 1025-word, 90° ROM may be expected to have 10-bit accuracy case, that is, where the sinusoids vary linearly. To solve Eq. (20), fix ℓ and m:

$$\varphi(\omega_k) = Y(\omega_k)\, X(\omega_k) e^{j\omega_k T}. \tag{21}$$

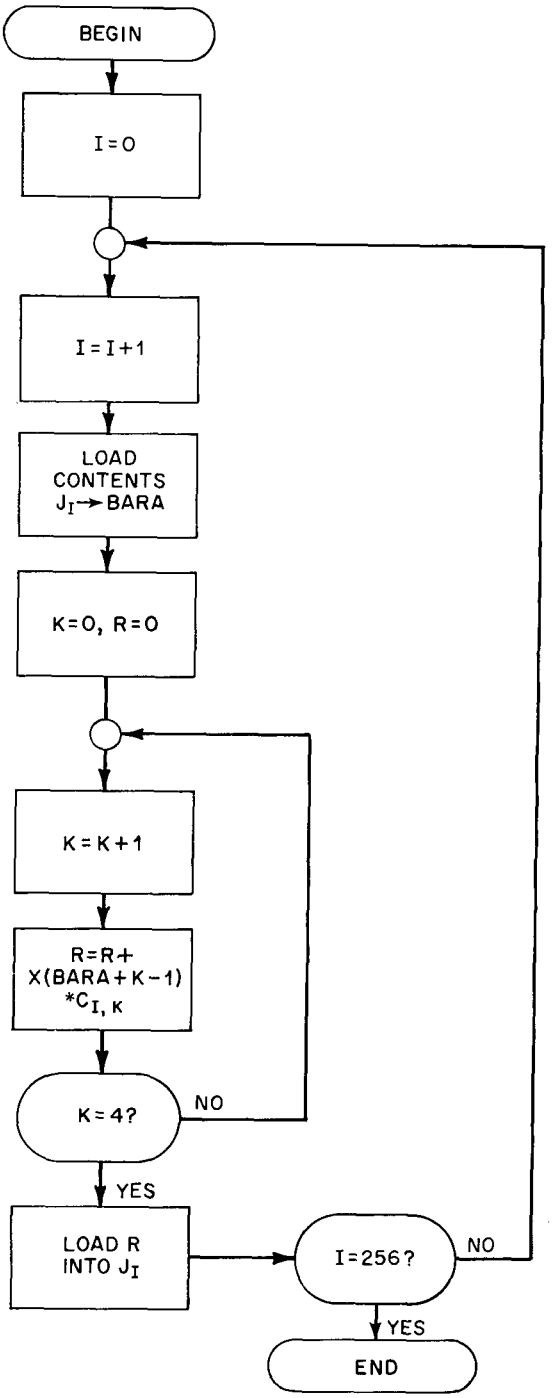


Fig. 11 — Flowchart of WGTSUM (I, K = indexes in Eq. (19); J_I = Ith entry in J table; R = register storage)

MLT is an array multiplication routine, and CONJG conjugates and circularly rotates data. Program macros and operation will be described.

Data Structure

Because of the number of macros involved, XCORREL has a large number of parameters in W store. Linkages are in $W(0)$, $W(10)$, and $W(11)$. $W(0)$ contains the LFFT address. The contents of $W(0)$ will be changed to the exit address in the course of execution. $W(10)$ contains the exit address, the jump address upon completion. $W(11)$ contains the address of CONJG. $W(3)$ and $W(4)$ contain the respective addresses of the X and Y arrays. $W(12)$ contains the rotation amount to be described. $W(1)$, $W(2)$, $W(5)$, $W(6)$, $W(8)$, and $W(9)$ are storage areas used by LFFT. The setup parameters for LFFT will, of course, specify a 1024-point transform.

Program Description

Arrays X and Y, located in BUFA and BUFB, respectively, input to the MLT routine where array multiplication is performed (see Fig. 12.) The resultant array is written over X. A DFT is performed on the resultant array using LFFT.

Transform both sides of Eq. (21) and use the convolution theorem:

$$\begin{aligned} f\{\varphi\} &= f\{YXej\omega_k T\} \\ &= f\{YX\} * f\{ej\omega_k T\} \end{aligned} \tag{22}$$

where * indicates convolution and $F\{\ \}$ indicates the Fourier transform of a discrete array. The transform of $ej\omega_k T$ is a Dirac δ function,

$$f\{\varphi\} = f\{YX(t-T)\}. \tag{23}$$

The argument $t-T$ indicates translation in time by T. Solve for φ by taking the inverse transform of both sides of Eq. (23). The inverse transform may be expressed as follows:

$$f^{-1}\{X(t)\} = \frac{\tilde{f}\{\tilde{X}\}}{N} \tag{24}$$

where the factor $1/N$ is provided for normalization, and the tilde indicates complex conjugation.

The program for evaluating φ relies on LFFT. LFFT is linked to the auxiliary programs MLT and CONJG. LFFT is used because it computes the DFT via the FFT. The resultant array is in the time domain. Then CONJG is used to conjugate and "rotate" the array. "Rotate" means location 1 of ARRAY 1 is written into location n of ARRAY 2, location 2 is written into location $n+1$, etc.; n is the rotation amount. After the last

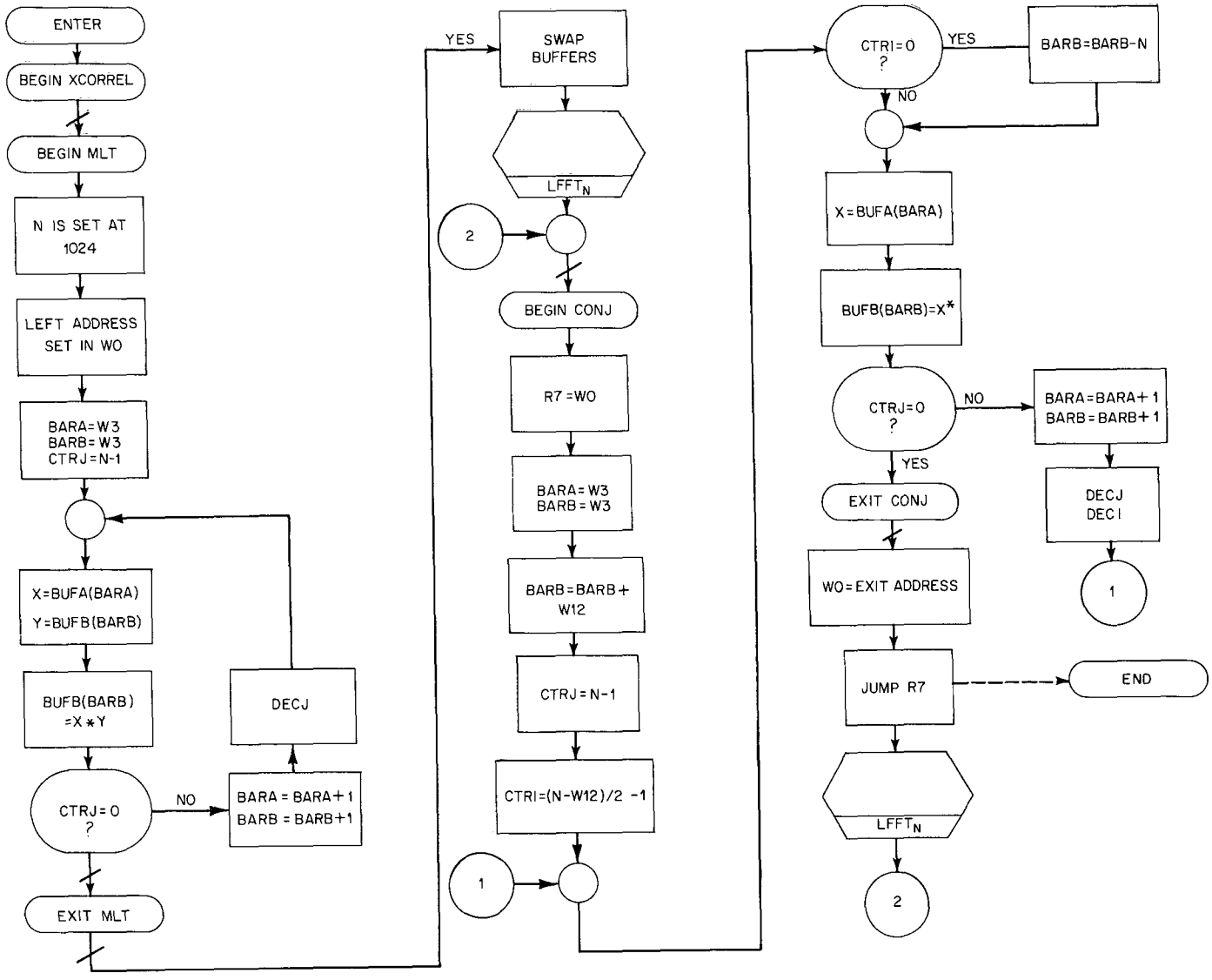


Fig. 12 — Flowchart of XCORREL. X and Y are local storage. "JUMP R7" is a switch set to exit upon entering CONJG the second time.

location (1024) of ARRAY 2 is filled, locations 1 to $n-1$ are filled; hence, a circular rotation of the data results. The circular rotation by n is a time translation by n where periodically repeating time series are assumed. A shift by n in the time domain series for a 1024-point transform produces a linear phase shift in the spectrum of $n/1024 = T$. The conjugation in CONJG is used to obtain the inverse transform as in Eq. (24). Another call to LFFT is made, and finally a call is made to CONJG with no circular rotation.

XCORREL, as it stands, is not very good computationally. The reason for this is that the macro LFFT used in the program is not normalized. If the forward DFT is defined as

$$F_m = \frac{1}{N} \sum_{k=0}^{n-1} Ske^{-j2\pi km/N}, \tag{25}$$

then the inverse should be

$$S_k = \sum_{m=0}^{N-1} F_m e^{j2\pi km/N}. \tag{26}$$

In this program, the factor $1/N$ also appears in the inverse transform definition. A change in SPAU design, the latching of the right shift of the input from BUFFER, makes possible proper scaling on the inverse DFT. This change was not present in the version of the SPAU simulator on which XCORREL was run, but was introduced in subsequent versions.

Performance

XCORREL was not run for 1024-point arrays. The number of clocks should be $(2 \log_2 N+4)N+140$ to within 30 clocks. For 1024 points, this would be 24,716 clocks, very expensive to run. For 128-point arrays, 2,418 clocks were used in a simulation.

SUMMARY OF SPAU PERFORMANCE

The LFFT and REFFT benchmarks were run for several different sizes of data arrays. These results, together with those of the other benchmarks, are summarized in Table 3. The elasped real time is shown based on an assumption of 150 ns for the period of one clock cycle.

Table 3
Time for SPAU Performance

Program	No. of Clocks	Time (ms)
LFFT (points):		
32	192	0.0288
64	423	0.0635
128	941	0.141
256	2,100	0.315
512	4,668	0.700
1024	10,307	1.547
4096	49,250	7.388
REFFT (points):		
64	365	0.0547
128	757	0.114
256	1,594	0.239
512	3,393	0.508
1024	7,241	1.088
OCT (1024 points)	16,422	2.46
WGTSUM (256 points)	1,543	0.225
Crosscorrelation:		
128 points, 1 beam, 1 channel	2,418	0.363
1024 points, 1 beam, 1 channel	24,716	3.707

DESIGN FEEDBACK

The coding of the benchmark programs gave rise to several significant changes in the design of the SPAU from that described in an earlier report [1]. The major changes were the following:

1. Changing the right-shift control on the channel inputs from a momentary instruction-by-instruction shift to a latched shift, which remains in effect indefinitely after it is set

2. Changing from a local overflow test on A5 to a global magnitude test so that potential overflows can be anticipated and the data can be scaled down to prevent overflow from occurring

3. Changing the *Y* store Read and Write control from increment to indirect addressing from the ROM Address Register (RAR), and applying this addressing to the *X* and *W* stores as well

4. Separating the buffer-identifying bits (the high-order bits) from the 12 low-order bits of each buffer address register so that buffer identifications may be swapped or held fixed while the low-order bits are incremented modulo 2^{12}

5. Adding two more adders, for a total of six, in the arithmetic section of the model XB-2 SPAU to support a radix-4 FFT algorithm. By making half as many buffer accesses, the radix-4 algorithm completes an FFT in half the time required by the radix 2 algorithm. Design changes 1 through 4 were reflected in the SPE software described in Ref. 8.

Other design changes were made, as may be observed by comparing the SPAU descriptions given in previous reports [1,9].

REFERENCES

1. W.R. Smith and H.H. Smith, "Signal Processing Element Functional Description," Part 2, (Preliminary) Signal Processing Arithmetic Unit," NRL Memorandum Report, 2522, Oct. 1972.

2. H.S. Elovitz, "Microprogrammed Benchmarks for the AN/UYK-17(XB-1) (V) Microprogrammed Control Unit (MCU)," NRL Report 7732, in preparation.

3. "What is the Fast Fourier Transform?," G-AE Subcommittee on Measurement Concepts, IEEE Trans. **AU-15**, No 2, (June 1967).

4. "On Digital Filtering," G-AE Concepts Subcommittee, IEEE Trans. **AU-16**, No. 3, 303-314 (Sept. 1968).

5. G.M. Jenkins and D.G. Watts, *Spectral Analysis and Its Applications*, Holden-Day, Inc., San Francisco, 1968.

6. M.C. Pease, "An Adaptation of the Fast Fourier Transform for Parallel Processing," J. Assoc. Comput. Mach. 15, 252-264 (Apr. 1968).

7. J.C. Stringer, "Fast Fourier Transform on the ASC,"TI ASC Internal Report, Job No. 2205-1, "Texas Instruments, Feb. 1973.

8. T.G. Rauscher, J.D. Roberts, Jr., and W.D. Elliott, "AN/UYK-17 Signal Processing Element Microcoding Support Software," NRL Report 7777, Nov. 1974.

9. W.R. Smith, J.P. Ihnat, H.H. Smith, N.M. Head, Jr., E. Freeman, Y.S. Wu, and B. Wald, "AN/UYK-17 (XB-1) (V) Signal Processing Element Architecture, NRL Report 7704, June 7, 1974.

Appendix A

BENCHMARK SPECIFICATIONS

INTRODUCTION

A group of processing functions has been selected as a representative sample of the operations to be carried out on the PROTEUS Analyzer and are presented here as benchmark problems. Each problem stresses one or more of the major analyzer capabilities (arithmetic operations, logic operations, and general data handling operations).

The results of these sample problems will be used in evaluating running times and overall capabilities of the proposed analyzer. In some instances the timing and storage performance for a particular problem will be used to project overall performance for that class of problem. The responses to these problems should be in a separate enclosure from the responses to the main body of the specification.

GUIDELINES

Problems should be implemented as if they were to be called from an overall executive routine or, if indicated, from some previous routine or via an interrupt. In either case, the implementation should include storage of "working registers" and any other overhead functions such as initializing tables, reformatting input or output data, and setting up I/O buffers.

The hardware and software configuraitons used for the benchmark problems must be identical to those proposed for the actual system, unless specified otherwise.

The solutions to the problems must be carried out using the problem organization that is defined. The contractor may redefine or redesign any of the problems (assuming the same tasks are accomplished) if the modification results in a specific advantage for his machine. The modifications must be documented in detail, and the resulting advantages must be demonstrated.

FUNCTION BENCHMARKS

Levels of Response

Two levels of response will be considered.

Minimum Required Response — The following response is required for each of the functional benchmark problems.

1. A detailed flow diagram of the program at the instruction level and a prose description of the solution

2. The overall problem execution time. Memory access time and individual instruction execution times should be itemized. Time required for any overhead functions shall be included.

3. Storage allocation in terms of total storage used and percentage of storage available for each of the program, data, coefficient and microprogram memories.

4. A symbolic instruction listing in the analyzer language with comments

5. A definition of each instruction (or each statement at each language level, if more than one level is used)

6. Execution times for each equation major loop or program segment

7. A detailed description of each special instruction or macro, including its purpose

8. Microprogram assembly and simulation, including (a) translation of the micro-instruction source language into the pattern of control bits to be used in a microprogram memory (All translator documentation outputs shall be provided.), and (b) simulation of the microprogram on a general-purpose computer. (This simulation should produce statistics on the execution of the microprogram, such as control memory size, timing, etc.)

9. Program assembly and simulation, including (a) assembly of the source code with all assembler outputs, and (b) simulation of the program on a general-purpose computer. (This simulation should produce statistics on the execution of the program, such as memory use and timing.)

10. The name or names of contractor personnel who can be contacted for answers to any questions of a technical nature regarding the method of problem solution.

Optional Responses — The contractor may wish to present his capabilities by executing the program on analyzer hardware, interfaced with a general-purpose computer to provide statistics and input/output capabilities. This response is not mandatory but is desirable to help evaluate the current state of system development.

Function Benchmark Problems

1. *Data Array Demultiplexing* — A 3200-point array consisting of 25 channels of time-multiplexed complex data (packed one complex sample per 32-bit memory word) (Fig. A1a) shall be demultiplexed into 25 arrays of 128 complex points each (Fig. A1b). The input data shall have previously been located in bulk memory, and the output shall result in 25 contiguous arrays in bulk memory.

2. *Data Array Demultiplexing* — Follow the steps of benchmark 1, assuming a 32,768-point input array consisting of 32 channels of time-multiplexed real data with 8-bit precision.

A. Input Array			B. Output Array		
Word	Channel	Sample	Word	Channel	Sample
1	1	1	1	1	1
2	2	2	2	1	2
3	3	1	3	1	3
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
25	25	1	128	1	128
26	1	2	129	2	1
27	2	2	130	2	2
28	3	2	131	2	3
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
50	25	2	256	2	128
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
3176	1	128	3073	25	1
3177	2	128	3074	25	2
3178	3	128	3075	25	3
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
3200	25	128	3200	25	128

Fig. A1 — Array demultiplexing input and output arrays

2. *Data Array Demultiplexing* — Follow the steps of benchmark 1, assuming a 32,768-point input array consisting of 32 channels of time-multiplexed real data with 8-bit precision.

3. *Complex Fast Fourier Transform* — The input data consists of 4096 complex values residing in bulk memory (packed one 32-bit word per complex sample). A 4096-point complex FFT shall be performed on the input data, and the results shall be returned to the same area of bulk memory that contained the input data.

4. *Real Fast Fourier Transform* — The input data consists of 1024-point real values, with 8-bit precision, residing in bulk memory. A 1024-point FFT shall be performed on the input data, and results shall be returned to a different area of bulk memory (packed one 32-bit word per complex sample).

5. *Data Array Demultiplexing and FFT* — Combine benchmarks 2 and 4 and perform FFT on 32 channels of time-multiplexed real data.

6. *Weighting Function With Irregular Addressing* — Apply a weighting function to an input data array using a predefined irregular addressing pattern on the input. This type of weighting function is used in forming constant-percentage resolution filters from constant-resolution FFT outputs. The input data will consist of an array of 1024 complex values formatted as shown in Fig. A2 and stored in bulk memory.

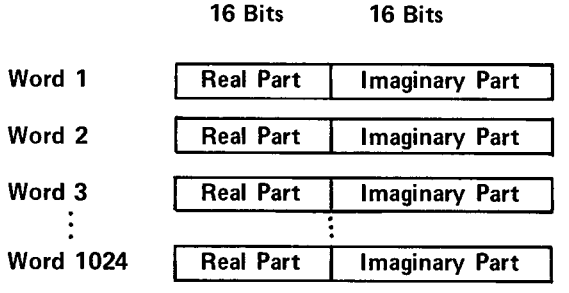


Fig. A2 — Input data array

The output will be an array of 512 complex values stored in bulk memory in the same format as the input data. The algorithm to be carried out is given by

$$Y_i = \sum_{k=1}^{4} C_{i,k}\, X_{J_i + k - 1} \qquad i = 1, 2, \ldots, 512$$

where Y_i is the ith complex output value.

The $C_{i,k}$ are the complex weighting factors stored in bulk memory as specified in Fig. A3.

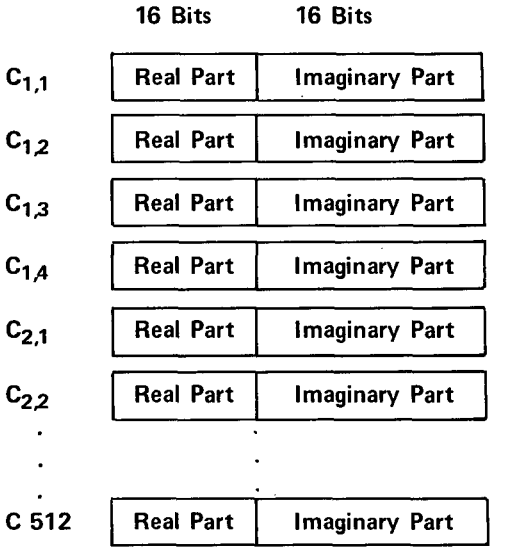


Fig. A3 — The $C_{i,k}$ weighting factors as start in bulk memory

The $X_{J_i + k - 1}$ are the four consecutive complex inputs used to form the ith output. J_i is defined in Table A1.

Table A1
Definitions XJ_i

i	J_i	i	J_i	i	J_i	i	J_i
1	1	23	30	45	61	67	93
2	2	24	31	46	63	68	95
3	3	25	32	47	65	69	96
4	4	26	34	48	66	70	97
5	6	27	35	49	68	71	99
6	7	28	37	50	69	72	100
7	8	29	38	51	70	73	102
8	9	30	39	52	72	74	103
9	10	31	42	53	73	75	105

Table A1 (Cont'd)

i	J_i	i	J_i	i	J_i	i	J_i
10	12	32	43	54	75	76	106
11	13	33	44	55	76	77	108
12	14	34	45	56	77	78	109
13	16	35	48	57	79	79	111
14	17	36	49	58	80	80	112
15	18	37	51	59	82	81	114
16	19	38	52	60	83	82	115
17	22	39	54	61	85	83	117
18	23	40	55	62	86	84	118
19	25	41	56	63	87	85	119
20	26	42	57	64	89	86	121
21	27	43	59	65	90	87	122
22	29	44	60	66	92	88	124
89	125	109	153	131	184	153	219
90	126	110	155	132	186	154	223
91	128	111	156	133	187	155	224
92	129	112	158	134	188	156	227
93	131	113	159	135	191	157	229
94	132	114	160	136	192	158	231
95	133	115	162	137	194	159	233
96	135	116	163	138	195	160	236
97	136	117	165	139	197	161	238

Table A1 (Cont'd)

i	J_i	i	J_i	i	J_i	i	J_i
98	138	118	166	140	199	162	241
99	139	119	167	141	201	163	243
100	140	120	169	142	202	164	246
101	142	121	170	143	204	165	248
102	143	122	172	144	205	166	250
103	145	123	173	145	207	167	253
104	146	124	174	146	209	16	255
105	147	125	176	147	210	169	258
106	149	126	177	148	211	170	261
107	151	127	178	149	213	171	265
108	152	128	180	150	215	172	267
175	276	129	181	151	216	173	269
176	279	130	183	152	218	174	273
177	281	197	343	219	407	241	466
178	284	198	345	220	409	242	469
179	287	199	348	221	413	243	471
180	291	200	352	222	416	244	473
181	294	201	356	223	419	245	476
182	297	202	358	224	421	246	478
183	299	203	361	225	424	247	480
184	303	204	364	226	426	248	483
185	307	205	367	227	429	249	485

Table A1 (Cont'd)

i	J_i	i	J_i	i	J_i	i	J_i
186	310	206	371	228	431	250	487
187	313	207	373	229	434	251	491
188	316	208	375	230	436	252	493
189	319	209	378	231	439	253	496
190	322	210	380	232	441	254	500
191	326	211	384	233	444	255	504
192	329	212	387	234	447	256	508
193	332	213	389	235	450		
194	335	214	392	236	454		
195	337	215	395	237	457		
196	339	216	399	238	459		
		217	403	239	461		
		218	405	240	463		

7. *Complex Demodulation and Octave Filtering* — The input will consist of a 1024-point real data array, packed as four 8-bit samples to a 32-bit memory word, residing in bulk memory (Fig. A4). The contractor will perform complex demodulation and octave filtering with decimation and scaling to form eight continguous 1-octave bands. A functional flowchart of the operations to be performed is presented in Fig. A5. All filters will be 4-pole, 4-zero recursive filters with nontrivial coefficients. The output shall be stored in bulk memory as eight separate arrays with one 32-bit complex output per word. All coefficients and results will be 16 bits for each of the real and imaginary parts.

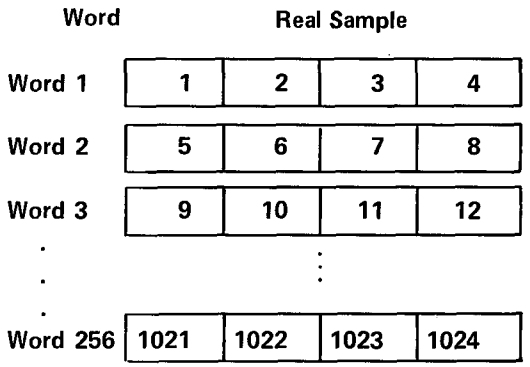


Fig. A4 — Contents of 1024-bit real data array

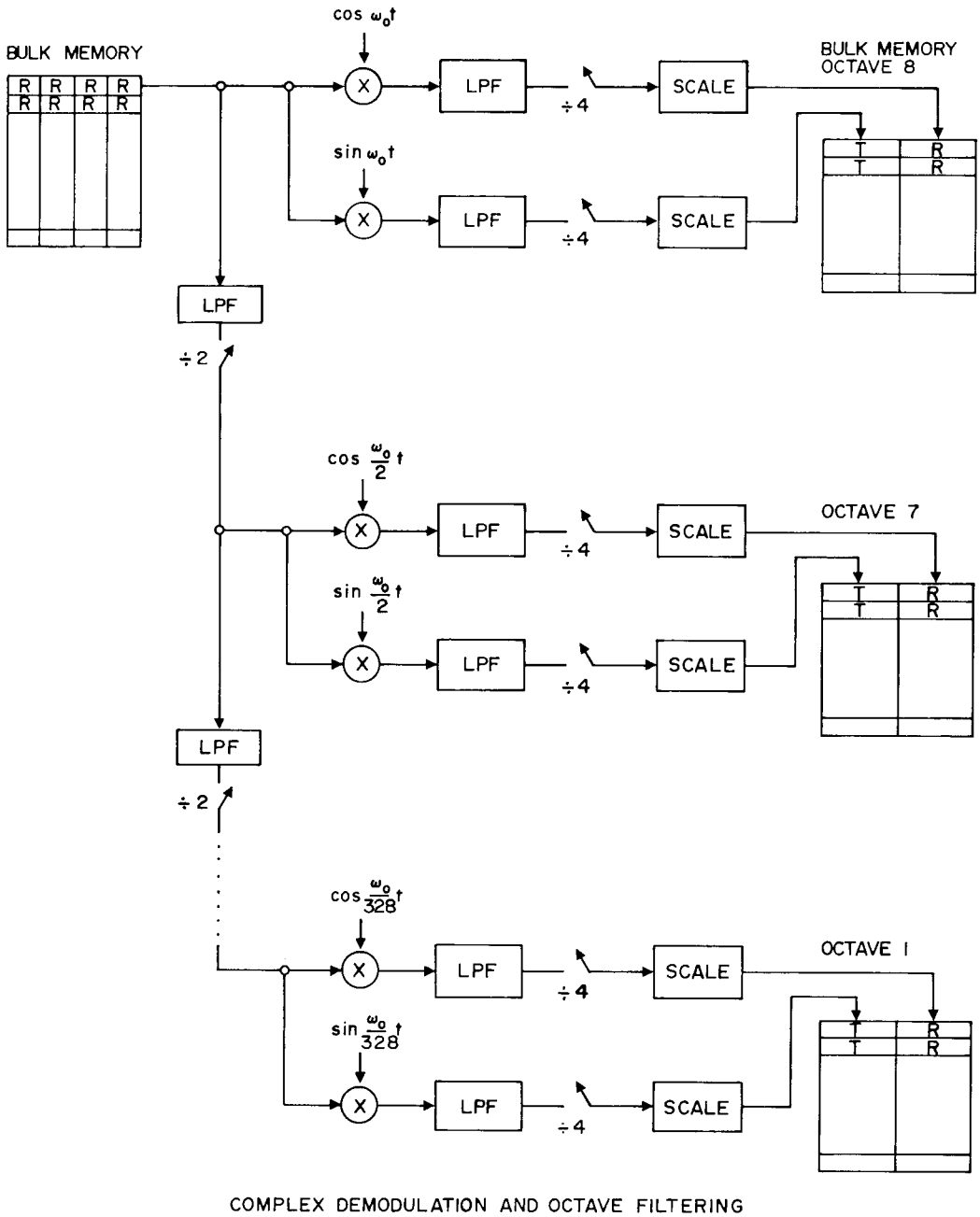


Fig. A5 — Complex demodulation and octave filtering

8. *Complex Demodulation and Octave Filtering* — Combine benchmarks 2 and 7 and perform complex demodulation and octave filtering on 32 channels of time-multiplexed real data.

9. *Demultiplexing, Octave Filtering, and FFT* — Follow the steps of benchmark 8 and then perform FFT on all channels and all octaves. The frequency domain output shall be stored in bulk memory packed one 32-bit complex output per word.

10. *Beam Forming* — Perform delay-sum beam forming on time-multiplexed input array as defined in benchmark 2, using the following formula:

$$Y_j\ (tk) = \frac{1}{32} \sum_{i=1}^{32} W_{ij} X_i(tk - T_{ij})$$

where

$X_i(tk)$ = ith input channel, Kth time sample

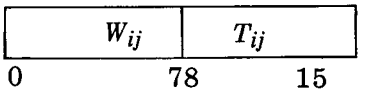
$Y_j(tk)$ = jth beam output, Kth time sample

T_{ij} = steering delay

W_{ij} = arbitrary scale factor

$i = 1, ..., 32$; $j = 1, ..., 32$; $K = 1, ..., 1024$.

W_{ij} and T_{ij} shall be a 1024-word (each word has 16 bits) stored table with 10-bit relative table address defined by ij such that the storage content of the table address ij has the following format:



The 32-beam output arrays shall be in contiguous time samples packed four 8-bit samples per 32-bit word and stored in bulk memory.

11. *Beam Forming and Octave Filtering* — Combine benchmarks 7 and 10 and perform complex demodulation and octave filtering on 32 beams.

12. *Beam Forming, Octave Filtering, and FFT* — Follow the steps of benchmark 11 and then perform FFT on all beams and all octaves. The frequency domain output shall be stored in bulk memory packed one 32-bit complex output per word.

13. *Sensor-Input, Beam-Output Cross-Correlation* — Compute the following cross-correlation function:

$$\Phi_{\ell m}(\omega_k) = Y_m(\omega_k) ej\omega_k T_{\ell m} X_\ell(\omega_k)$$

where

$Y_m(\omega_k)$ = mth beam, Kth frequency

$ej\omega_k T_{\ell m}$ = complex spatial term with respect to $T_{\ell m}$

$T_{\ell m}$ = steering delay

$X_\ell(\omega_k)$ = ℓth input channel, Kth frequency sample

$\ell = 1, ..., 32; \quad m = 1, ..., 32; \quad K = 1, ..., 1024.$

(X_ℓ) and (Y_ℓ) are packed, 32-bit complex words stored in the bulk memory. Computations of $\{ej\omega_k T_{\ell m}\}$ should be carried in at least 8-bit real and imaginary parts of a complex number. $(\Pi\Phi m)$ output shall be stored in bulk memory packed in 16-bit words (8-bit real and imaginary). It is acknowledged that the proposed basic analyzer unit configuration may not have sufficient bulk storage to execute this benchmark program. However, computer simulation of this program should adequately demonstrate the bulk memory expansion capability, dynamic paging and overlaying between bulk memory and the analyzer unit, and overall system throughput.

14. *Gram Thresholding and Formatting* — The input will be an array of 256 real data points (specified in Table 2) representing the outputs of the STI processor, as specified. The input data shall be stored in bulk memory as two 16-bit values per data word. The problem is to quantize the STI outputs by computing a moving window noise mean for each input data point and use the noise mean to threshold the data to a 2-bit amplitude. The algorithm to be used is defined in the specification, with the following parameter definitions:

$W = 32$

K_{cs} = (specified by user)

K_{rs} = (specified by user)

$N = 256$

K_1 = (specified by user)

K_2 = (specified by user)

K_3 = . (specified by user)

The output shall be 128 values packed as sixteen 2-bit amplitudes formatted into eight 32-bit words and returned to bulk memory as shown in Fig. A6.

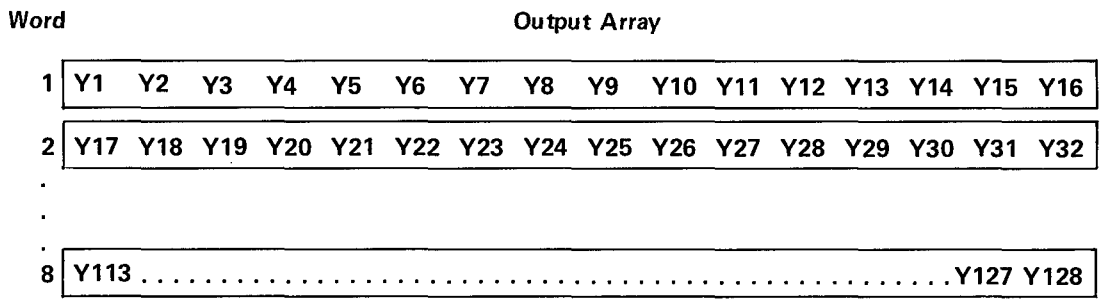


Fig. A6 — Output array for gram thresholding and formatting

15. *Bearing Computation (DIFAR)* — The input to the bearing computation shall consist of two arrays, each of 1024 points, in bulk memory, representing 2048 points of the sine and cosine channels, respectively, after LTI. The format of the input data shall be as in Fig. A7. The algorithm shall be as specified. The output shall be a 512-point array consisting of four 8-bit bearing outputs per 32-bit word in bulk memory. The format of the output shall be as shown in Fig. A8.

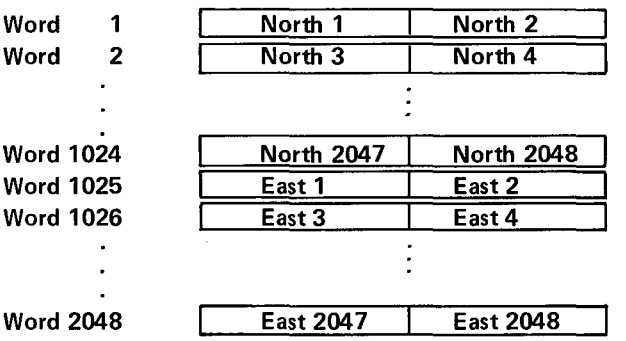


Fig. A7 — Input format for bearing calculation

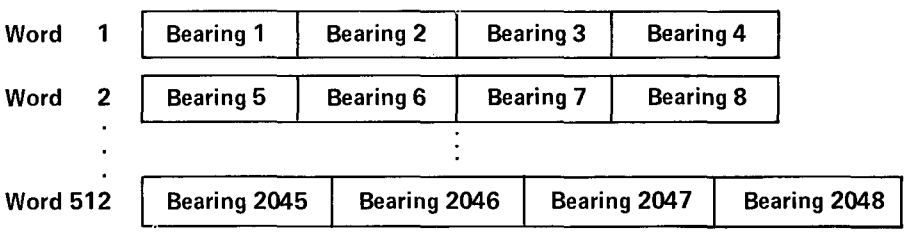


Fig. A8 — Output format for bearing calculation

GLOSSARY

ACSR - Alternate Control Store Address Register

AMIL - A Microprogramming Language

BSM - Buffer Storage Modules

FFT - Fast Fourier Transform

FSCR - Field Select Control Register

FSDR - Field Select Data Register

FSU - Field Select Unit

MCU - Microprogrammed Control Unit

SCC - Selector Channel Controller

SPAU - Signal Processing Arithmetic Unit

SPE - Signal Processing Element

Appendix B

PROGRAM LISTINGS

SPAU MICROPROGRAMMING LANGUAGE

The program listings that follow are expressed in a symbolic language dubbed ANIMIL (ANother Interesting MIcroprogramming Language).† Individual microinstructions may occupy several lines on the listing; blanks and the ends of physical lines have no logical significance. Each microinstruction, which may be labeled to facilitate symbolic addressing, consists of subcommands that reference specific SPAU facilities and indicate actions the SPAU is to take. A sharp or number sign (#) must be used to separate a label from its microinstruction. The end of a microinstruction is indicated by a dollar sign. Subcommands are separated by semicolons, and they may appear in any order. Comments are strings of characters, including physical ends of lines, which are initiated and terminated by double quotation marks. Comments may appear anywhere except within comments.

REGISTERS

The following is a list of the single registers:

ACSAR	P4
BARA	R1
BARB	R2
CTRI	R3
CTRJ	R4
CTRK	R7
INCA	RAR
INCB	Z1
INCR	Z2
P1	Z3
P2	Z4
P3	

The array registers are COND (subscript), W (subscript), X1 (subscript), X2 (subscript), Y1 (subscript), and Y2 (subscript). The subscript for array registers other than the W array may range from 0 to 15. The W array has a range of 0 to 31. Ranges of

†T.G. Rauscher, J.D. Roberts, Jr., and W.D. Elliott, "AN/UYK-17(XB-1) (V) Signal Processing Element Microcoding Support Software," NRL Report 7777, Nov. 1974.

subscripts for array registers may be written as follows:

subscript - subscript

For example, X1(3-5). The elements of the COND array correspond to 16 SPAU conditions for branching.

BUFA and BUFB (half word number [-half word number]) are two memory registers. Buffer references are to half words (16 bits), not to full words (32 bits). Even-numbered half words starting with zero refer to the low-order 16 bits in consecutive words. The half words in each buffer are referenced 0-2047. The other memory register is ROM (word number [-word number]). ROM word numbers range from 0 to 2047. The first 1025 ROM words contain two 16-bit coefficients that are the sine and cosine of angles between 0 and $\pi/2$ at intervals of $\pi/2048$. These sine and cosine coefficients may not be altered. When read into the Z registers, the first (sine) coefficient is put into Z1 and the second (cosine) coefficient is put into Z2. The remaining 1023 coefficient store words contain four 16-bit coefficients that may be changed under microprogram control. When read into Z registers, the leftmost (first) coefficient is put into Z1, the second coefficient is put into Z2, etc. Unlike other facilities, each reference to ROM indicates either two values (if the address is less than or equal to 1025) or four values (if the address is greater than 1025).

SPAU PROGRAMS

ANIMIL listings* for benchmark programs are given in the following pages.

*T.G. Rauscher, J.D. Roberts, Jr., and W.D. Elliott, "AN/UYK-17(XB-1) (V) Signal Processing Element Microcoding Support Software," NRL Report 7777, Nov. 1974.

```
 73/03/07. 15.46.01.
PROGRAM   LFFT

     "   SPAU PROGRAM    'LFFT'

   W(1)=N/2  W(2)=LOG2(N)  W(3)= DATA ADDRESS  W(4)= RESULT ADDRESS
   W(5)&CTRK=BUTTERFLIES PER HALF GROUP
   W(6)&CTRJ=GROUPS PER STAGE
     CTRI=STAGE      W(8)=ROM INCREMENT  W(9)=1  "


L0# A6:BARB=INCB; A7:RAR=LIT+INCR; A5:INCA=W(1); CTRJ=W(1)$

L1# A5:BARA=W(3); A6:INCB=BARB-INCB; A7:RAR=RAR-INCR$

L2# A5:BARA=BARA+INCA; A6:INCB=LIT+INCB; LIT=1; CTRK=LIT; W(5)=A6$

L3# A5:BARA=BARA-INCA; X(1)=RSHBUFA; Z=ROM$
    "READ ROM ONLY THIS ONCE IN STAGE 1."

L4# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:INCB=LIT-INCB; Y(1)=RSHBUFA;
 W(6)=CTRJ$
    "START STAGE 1 PIPELINE HERE."

L5# M1:X2(1)*Z1; A5:BARA=BARA-INCA; X(2)=RSHBUFA; CTRI=W(2); DECJ$

L6# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=W(4); Y(2)=RSHBUFA; DECJ;
 A1:R1=P1+P2; A2:R2=Y1(1)+A1; A3:R3=P3-P4; A4:R4=Y2(1)+A3$

L7# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; A7:LIT;
 LIT=1; X(1)=RSHBUFA; BUFB=R2R4; A1:R1=Y1(1)-R1; A3:R3=Y2(1)-R3;
 DECJ; W(9)=A7$
    "STAGE 1 LOOPS HERE FROM L10."

L8# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB-INCB; Y(1)=RSHBUFA;
 BUFB=R1R3; A1:R1=P1+P2; A2:R2=Y1(2)+A1; A3:R3=P3-P4;
 A4:R4=Y2(2)+A3$

L9# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; X(2)=RSHBUFA;
 BUFB=R2R4; A1:R1=Y1(2)-R1; A3:R3=Y2(2)-R3; DECJ; A7:INCR=LIT;
 LIT=-1024; W(8)=A7$
    "SET INCR TO BE ABLE TO READ ROM AT 0 DEGREES (RAR=0)
      AND AT 180 DEGREES (RAR=1024) IN STAGE 2."

L10# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB-INCB; Y(2)=RSHBUFA;
 BUFB=R1R3; A1:R1=P1+P2; A2:R2=Y1(1)+A1; A3:R3=P3-P4;
 A4:R4=Y2(1)+A3; IF NOT CTRJ THEN GO TO L7$

L11# M1:X2(2)*Z1; A5:BARA=W(4); A6:BARB=BARB-INCB; BUFB=R2R4;
 A1:R1=Y1(1)-R1; A3:R3=Y2(1)-R3$
```

```
L12# A5:BARA=BARA+INCA; A6:BARB=BARB-INCB; A7:LIT+RSHW(6); BUFB=R1R3;
 A1:R1=P1+P2; A2:R2=Y1(2)+A1; A3:R3=P3-P4; A4:R4=Y2(2)+A3; W(6)=A7;
 CTRJ=RSHW(6)$

L13# A6:BARB=BARB-INCB; BUFB=R2R4; A1:R1=Y1(2)-R1; A3:R3=Y2(2)-R3;
 DECI$

L14# BUFB=R1R3; SWAP$

L15# A5:BARA=BARA-INCA; A6:INCB=LIT-LSHW(5); X(1)=RSHBUFA; Z=ROM$

L16# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A7:RAR=RAR-INCR;
 Y(1)=RSHBUFA$
    "START STAGE 2 PIPELINE HERE."

L17# M1:X2(1)*Z1; A5:BARA=BARA-INCA; X(2)=RSHBUFA;
 Z=ROM$

L18# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=W(4);
 A7:RAR=RAR+INCR; Y(2)=RSHBUFA; A1:R1=P1+P2; A3:R3=P3-P4;
 A2:R2=Y1(1)+A1; A4:R4=Y2(1)+A3; DECJ$

L19# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; X(1)=RSHBUFA;
 BUFB=R2R4; Z=ROM; A1:R1=Y1(1)-R1; A3:R3=Y2(1)-R3$
    "STAGE 2 LOOPS HERE FROM L22."

L20# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR-INCR; Y(1)=RSHBUFA; BUFB=R1R3; A1:R1=P1+P2; A2:R2=Y1(2)+A1;
 A3:R3=P3-P4; A4:R4=Y2(2)+A3$
    "INCREMENT RAR UP TO 1024."

L21# M1:X2(1)*Z1: A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; X(2)=RSHBUFA;
BUFB=R2R4; A1:R1=Y1(2)-R1; A3:R3=Y2(2)-R3;DECJ; Z=ROM$

L22# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+W(9); Y(2)=RSHBUFA;
 A7:RAR=RAR+INCR; BUFB=R1R3; A1:R1=P1+P2; A2:R2=Y1(1)+A1;
 A3:R3=P3-P4; A4:R4=Y2(1)+A3; IF NOT CTRJ THEN GO TO L19$
    "DECREMENT RAR BACK TO 0."

L23# M1:X2(2)*Z1; A5:BARA=W(4); A6:BARB=BARB-INCB; BUFB=R2R4;
 A1:R1=Y1(1)-R1; A3:R3=Y2(1)-R3$

L24# A6:BARB=BARB+INCB+1; A5:LIT+RSHW(6); CTRJ=RSHW(6); W(6)=A5;
 BUFB=R1R3; A1:R1=P1+P2; A2:R2=Y1(2)+A1; A3:R3=P3-P4; A4:R4=Y2(2)+A3$

L25# A6:BARB=BARB-INCB;
 BUFB=R2R4; DECI; A1:R1=Y1(2)-R1; A3:R3=Y2(2)-R3$

L26# A5:BARA=BARA+INCA; A6:BARB=LIT+INCB; W(5)=A7; BUFB=R1R3;
 CTRK=LSHW(5); SWAP; A7:LIT+LSHW(5)$
```

```
L27# A5:BARA=BARA-INCA; A6:INCB=BARB+INCB; X(1)=RSHBUFA; Z=ROM;
 A7:INCR=LIT+RSHW(8); W(8)=A7$

L28# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=LIT+W(4);
 A7:RAR=RAR-INCR; Y(1)=RSHBUFA; LIT=-1; DECK$
    "START GENERAL STAGE HERE."

L29# M1:X2(1)*Z1; A5:BARA=BARA-INCA; X(2)=RSHBUFA; Z=ROM; DECK$

L30# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A7:RAR=RAR-INCR; Y(2)=RSHBUFA;
 A1:R1=P1+P2; A2:R2=Y1(1)+A1; A3:R3=P3-P4; A4:R4=Y2(1)+A3; IF CTRK
 THEN GO TO L5Ø; A6:BARB=BARB+W(9)$
    "IF CTRK=Ø THEN THIS IS STAGE 3 (THERE ARE TWO
      BUTTERFLIES PER HALF GROUP)."

L31# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB;
 X(1)=RSHBUFA; BUFB=R2R4; Z=ROM; A1:R1=Y1(1)-R1; A3:R3=Y2(1)-R3$
    "ACUTE ANGLE (FIRST) HALF OF GENERAL STAGE LOOPS
      HERE FROM L34."

L32# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR-INCR; Y(1)=RSHBUFA; BUFB=R1R3; DECK; A1:R1=P1+P2;
 A2:R2=Y1(2)+A1; A3:R3=P3-P4; A4:R4=Y2(2)+A3$

L33# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB;
 X(2)=RSHBUFA; BUFB=R2R4; DECK; Z=ROM; A1:R1=Y1(2)-R1;
 A3:R3=Y2(2)-R3$

L34# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR-INCR; Y(2)=RSHBUFA; BUFB=R1R3; A1:R1=P1+P2;
 A2:R2=Y1(1)+A1; A3:R3=P3-P4; A4:R4=Y2(1)+A3; IF NOT CTRK
 THEN GO TO L31$

L35# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB;
 X(1)=RSHBUFA; BUFB=R2R4; CTRK=W(5); Z=ROM; A1:R1=Y1(1)-R1;
 A3:R3=Y2(1)-R3$
    "BEGIN TRANSIT TO SECOND HALF OF GENERAL STAGE."

L36# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1;
 A6:BARB=BARB+INCB+1; A7:RAR=RAR+INCR; Y(1)=RSHBUFA;
 BUFB=R1R3; DECK; A1:R1=P1+P2; A2:R2=Y1(2)+A1; A3:R3=P3-P4;
 A4:R4=Y2(2)+A3$

L37# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB;
 X(2)=RSHBUFA; BUFB=R2R4; DECK; Z=ROM; A1:R1=Y1(2)-R1;
 A3:R3=Y2(2)-R3$

L38# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1;
 A6:BARB=BARB+INCB+1; A7:RAR=RAR+INCR; Y(2)=RSHBUFA; BUFB=R1R3;
 A1:R1=P1-P2; A2:R2=Y1(1)+A1; A3:R3=P3+P4; A4:R4=Y2(1)-A3$
```

```
L39# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; DECK;
 X(1)=RSHBUFA; BUFB=R2R4; Z=ROM; A1:R1=Y1(1)-R1; A3:R3=Y2(1)+R3$
    "OBTUSE ANGLE (SECOND) HALF OF GENERAL STAGE LOOPS
      HERE FROM L42."

L40# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR+INCR; Y(1)=RSHBUFA; BUFB=R1R3;
 A1:R1=P1-P2; A2:R2=Y1(2)+A1; A3:R3=P3+P4; A4:R4=Y2(2)-A3$

L41# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB;
 X(2)=RSHBUFA; BUFB=R2R4; DECK; Z=ROM; A1:R1=Y1(2)-R1;
 A3:R3=Y2(2)+R3$

L42# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 BUFB=R1R3; A1:R1=P1-P2; A2:R2=Y1(1)+A1; A3:R3=P3+P4;
 A4:R4=Y2(1)-A3; A7:RAR=RAR+INCR; IF NOT CTRK THEN GO TO L39;
 Y(2)=RSHBUFA$

L43# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; Z=ROM;
 X(1)=RSHBUFA; BUFB=R2R4; CTRK=W(5); DECJ; A1:R1=Y1(1)-R1;
 A3:R3=Y2(1)+R3$

L44# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR-INCR; Y(1)=RSHBUFA; BUFB=R1R3; A1:R1=P1-P2;
 A2:R2=Y1(2)+A1; A3:R3=P3+P4; A4:R4=Y2(2)-A3; DECK$

L45# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; Z=ROM;
 BUFB=R2R4; A1:R1=Y1(2)-R1; A3:R3=Y2(2)+R3; DECK; X(2)=RSHBUFA;
 ACSAR=LIT; LIT=L59$

L46# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; Y(2)=RSHBUFA; A7:RAR=RAR-INCR;
 A1:R1=P1+P2; A2:R2=Y1(1)+A1; A3:R3=P3-P4; A4:R4=Y2(1)+A3;BUFB=R1R3;
 A6:BARB=BARB+W(9); IF NOT CTRJ THEN GO TO L31$

L47# A5:BARA=W(4); A7:RAR=LIT; DECI; SWAP$

L48# A5:BARA=BARA+INCA; A6:RSHW(6); W(6)=A6; CTRJ=RSHW(6); IF
 CTRI THEN GO TO ACSAR; A7:R7=LIT; LIT=L27$
    "THE FIRST TIME HERE ACSAR CONTAINS L48 IT WILL BE
      RESET LATER AT L45."

L49# A5:LSHW(5); W(5)=A5; CTRK=LSHW(5); GO TO R7; A6:BARB=LIT+INCB$
    "STAGE 3 CONTINUES HERE, AND ALSO LOOPS HERE
      FROM L57."

L50# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; Z=ROM;
 X(1)=RSHBUFA; BUFB=R2R4; A1:R1=Y1(1)-R1; A3:R3=Y2(1)-R3$
```

```
L51# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR+INCR;
 Y(1)=RSHBUFA; BUFB=R1R3; A1:R1=P1+P2; A2:R2=Y1(2)+A1;
 A3:R3=P3-P4; A4:R4=Y2(2)+A3$

L52# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; Z=ROM;
 X(2)=RSHBUFA; BUFB=R2R4; A1:R1=Y1(2)-R1; A3:R3=Y2(2)-R3$

L53# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR+INCR; Y(2)=RSHBUFA; BUFB=R1R3; A1:R1=P1-P2;
 A2:R2=Y1(1)+A1; A3:R3=P3+P4; A4:R4=Y2(1)-A3; DECJ$
    "CHANGE SIGN OF P2, P4, AND A3 IN ORDER TO
      ACCOMMODATE 135 DEGREES."

L54# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; Z=ROM;
 X(1)=RSHBUFA; BUFB=R2R4; A1:R1=Y1(1)-R1; A3:R3=Y2(1)+R3$

L55# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR-INCR; Y(1)=RSHBUFA; BUFB=R1R3; A1:R1=P1-P2;
 A2:R2=Y1(2)+A1; A3:R3=P3+P4; A4:R4=Y2(2)-A3; ACSAR=LIT; LIT=L48$

L56# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; Z=ROM;
 X(2)=RSHBUFA; BUFB=R2R4; A1:R1=Y1(2)-R1; A3:R3=Y2(2)+R3$

L57# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A7:RAR=RAR-INCR; Y(2)=RSHBUFA;
 A1:R1=P1+P2; A2:R2=Y1(1)+A1; A3:R3=P3-P4; A4:R4=Y2(1)+A3; BUFB=R1R3;
 A6:BARB=BARB+W(9); IF NOT CTRJ THEN GO TO L50$

L58# A5:BARA=W(4); A7:RAR=LIT; DECI;
 SWAP; GO TO ACSAR$
    "END OF STAGE 3. ENTER GENERAL STAGE VIA L48."

L59# INTERRUPT$
```

```
"                    REFFT:
                                A BACKEND MACRO FOR THE FFT WHICH EF-
FICIENTLY PROCESSES REAL SERIES.  A REAL VECTOR IS PACKED INTO A
VECTOR OF LENGTH N/2 BY PLACING EVEN INDEXED COMPONENTS IN THE REAL
PART AND ODD INDEXED COMPONENTS IN THE IMAGINARY.  AN FFT OF
THIS N/2 VECTOR IS TAKEN, THE RESULT IS FED INTO THE REFFT
PROGRAM.  REFFT UNPACKS THE DATA AND PERFORMS THE LAST STAGE OF THE
N-POINT FFT THEN OUTPUTS INTO THE APPROPRIATE LOCATIONS.
     W(8) CONTAINS THE ROM INCREMENT -(4096/N),N=#REAL PTS.
     W(3) CONTAINS THE STARTING ADDRESS AND FINAL ADDRESS.
     W(4) CONTAINS ADDRESS OF WORK-SPACE IN BUFB.
     W(1) CONTAINS N/4, CONSISTENT WITH N/2 PT. COMPLEX FFT.
     OTHER VALUES FOLLOW FFT FORMAT.
     Z IS ASSUMED EQUAL (SIN,COS); FORWARD TRANSFORM IS EXP(-J0)
      LOOP ADDRESSING: K IS INDEX THEN BARA ADDRESSES POINTS K,N-K
      BARB ADDRESSES POINTS K,K+N/2
"

RFT0#  LIT=0; Y2(4)=LIT; A5: BOTH=W(3); A6: BARB=W(3)$"INITIALIZE
          ADDRESSES"

RFT1#  Y(2)=RSHBUFA; A5: BARA=LIT+INCA+1; LIT=0; W(6)=A5; A3: R4=
          Y2(4)+Y2(4)$"INPUT PT. 0, TREAT AS SPECIAL CASE"

RFT2#  X(1)=RSHBUFA; A5: BARA=INCA; CTRJ=W(1); A2: R2=Y1(2)-
          Y2(2); A3: R3=Y1(2)+Y2(2)$"HAVE FORMED 0 & N/2 FREQ. PTS.
          IN R3R4,R2R4; SET CTRJ FOR FST. LOOP"

RFT3#  A5: BARA=BARA+INCA; DECJ;"DECJ FOR PREVIOUS X(1) READ" LIT=0;
          A7: INCR=LIT-RSHW(8)$"LOAD ROM INCREMENT"

RFT4#  A5: BOTH=BARA+LSHW(1); A6: INCB=LIT+LSHW(1); LIT=4095$
          "2*(ST. ADDRESS)+N/2 TO INCA, N/2-1 TO INCB"

RFT5#  A5: BARA=BARA-W(6)$"SET ADDRESS FOR INPUT PT. N/2-1"

RFT6# A6: BARB=W(4)$

RFT7#  A7: RAR=LIT+INCR; BUFB=R3R4; A6: BARB=BARB+INCB+1;"OUTPUT
          0-RESULT, SET FOR N/2" Y(1)=RSHBUFA; A5: BARA=LIT+W(6)+1;
          LIT=0; W(6)=A5$

RFT8#  LIT=LP1; ACSAR=LIT; Z=ROM; X(2)=RSHBUFA; DECJ; A5: BARA-INCA;
          W(7)=A5$

RFT9#  A5: BARA=LIT-W(7); LIT=0;  A1: Y1(3)=X1(1)+Y1(1); A2:
          Y2(3)=X2(1)-Y2(1); A3: X1(3)=Y2(1)+X2(1); A4: X2(3)=
          Y1(1)-X1(1)$

RFT10#  LIT=0; Y(2)=RSHBUFA; W(6)=A5; A5: BARA=LIT+W(6)+1$

RFT11#  X(1)=RSHBUFA; DECJ; A5: BARA-INCA; W(7)=A5; M1: X2(3)*Z1;
          M2: X1(3)*Z2; M3: X2(3)*Z2; M4: X1(3)*Z1$

 LP1# A7:  RAR=RAR+INCR; "OUTPUT N/2+2J RESULT" BUFB=R2 R4; A1:
Y1(5)=X1( 2)+Y1(2); A2:  Y2(5)=X2(2)-Y2(2); A5:  BARA=LIT-W(7); LIT=0;
A6:BARB=B ARB-INCB; A3:  X1(5)=Y2(2)+X2(2); A4:  X2(5)=Y1(2)-X1(2)$
```

```
LP2# Y(1)=RSHBUFA; A5:  BARA=LIT+W(6)+1; W(6)=A5; Z=ROM; A1:  R1=P 1+P2;
A2:  R2=Y1(3)+A1;
 A3:  R3=P3-P4 ; A4:  R4=Y2(3)+A3; $

LP3# LIT=0; X(2)=RSHBUFA;DECJ; BUFB=R2R4; "OUTPUT 2J+1 RESULT" W(7)=A5;
A5:  BA RA-INCA;
 A1:  R 2=Y1(3)-R1; A3:  R4=Y2(3)-R3; A6:  BARB=BARB+INCB+1;
M1:X2(5)*Z1; M2:  X1(5)*Z2; M3:  X2(5)*Z2; M4:  X1(5)*Z1$

LP4# A5:  BARA=LIT-W(7); A7:  RAR=RAR+INCR; "OUTPUT N/2+2J+1 RESULT"
BUFB=R2R4; A1:Y1(3)=X1(1 )+Y1(1); A2:  Y2(3 )=X2(1)-Y2(1); A3:  X1(3)=
Y2(1)+X2(1); A4:  X2(3)=Y1(1)-X1(1); A6:  BARB=BARB-INCB$

LP5# W(6)=A5; Y(2)=RSHBUFA; A5:
BARA=LIT+W( 6)+1; LIT=0 ; A1:  R1=P1+P2;
 A2:  R2= Y1(5)+A1; A3:  R3=P3-P4; A4:  R4=Y2(5)+A3; Z=ROM$

LP6# IF NOT CTRJ THEN GO TO ACSAR; DECJ;  "OUTPUT 2J RESULT" BUFB=R2R4;
X
(1)=RSHBUFA; A5:  BARA-INCA;
 W(7)=A 5;
   A1:  R2=Y1(5)-R1; A3:  R4=Y2(5)-R3; A6:  BARB=B ARB+INCB+1; M1:
X2(3)*Z1; M2:X1(3)*Z2; M3:X2(3)*Z2; M4:  X1(3)*Z1$

"  NEXT SECTION KEEPS PIPE FULL AND PERFORMS BONUS MULTIPLY BY J
         THE NEXT SECTION WILL STEP DOWN THRU THE ROM AND COMPUTE COMPX
         PRODUCTS AS THOUGH ROM COEFFICIENTS RANGED FROM PI/2 TO PI.
         NEXT LOOP WILL YIELD RESULTS CONJUGATE TO THOSE IN FIRST LOOP
"
RFT18#  LIT=NLP0; ACSAR=LIT; CTRJ=W(1); A1: X1(5)=Y1(2)+Y1(2);
         A2: Y2(5)=X2(2)-Y2(2); A4: Y1(5)=Y1(2)+X1(2); A6: BARB=BARB-
         INCB; BUFB=R2R4$


RFT19#  DECJ; LIT=0; A1: R1=P1+P2; A2: R2=Y1(3)+A1; A3: R3=P3-P4;
         A4: R4=Y2(3)+A3; A5: BARA=LIT-W(7)$

RFT20#  LIT=0; Y(1)=RSHBUFA; BUFB=R2R4; A1: R2=Y1(3)-R1; W(6)=A5;
         A3: R4=Y2(3)-R3; A5: BARA=LIT+W(6)+1; A6: BARB=BARB+INCB+1$

RFT21#  A3: R3=Y2(2)+X2(2); X(2)=RSHBUFA; DECJ; BUFB=R2R4; A5: BARA-INCA
         W(7)=A5; A6: BARB=BARB-INCB$

RFT22#  A1: R1=X1(5)-Y1(5); A2: R2=Y1(5)+A1; A3: R4=Y2(5)-R3;
         A5: BARA=LIT-W(7); LIT=0$

RFT23#  Y(2)=RSHBUFA; W(6)=A5; A5: BARA=LIT+W(6)+1; LIT=0; DECJ$

RFT24#  A1: Y1(3)=X1(1)+Y1(1); A2: Y2(3)=X2(1)-Y2(1); A3: X1(3)= Y2(1)
         +X2(1); A4: X2(3)=Y1(1)-X1(1); GO TO NLP3$

NLP0#  X(2)=RSHBUFA; DECJ; A5: BARA-INCA; W(7)=A5; BUFB=R2R4; A1:
         R2=Y1(3)-R1; A3: R4=Y2(3)+R3; A6: BARB=BARB+INCB+1; M1:
         X2(5)*Z1; M2: X1(5)*Z2; M3: X2(5)*Z2; M4: X1(5)*Z1$

NLP1#  A5: BARA=LIT-W(7); LIT=0; BUFB=R2R4; A1: Y1(3)=X1(1)+Y1(1); A2:
         Y2(3)=X2(1)-Y2(1); A3: X1(3)=Y2(1)+X2(1); A4: X2(3)=
         Y1(1)-X1(1); A6: BARB=BARB-INCB; A7: RAR=RAR-INCR$

NLP2#  Z=ROM; Y(2)=RSHBUFA; W(6)=A5; A5: BARA=LIT+W(6)+1; LIT=0;
         A1:R1=P1-P2; A2: R2=Y1(5)+A1; A3: R3=P3+P4; A4: R4=
         Y2(5)-A3$
```

```
NLP3#  DECJ; X(1)=RSHBUFA; A5: BARA-INCA; W(7)=A5; BUFB=R2R4; A6:
          BARB=BARB+INCB+1; M1: X2(3)*Z1; M2: X1(3)*Z2; M3: X2(3)*
          Z2; M4: X1(3)*Z1; A1: R2=Y1(5)-R1; A3: R4=Y2(5)+R3$

 NLP4# A7:  RAR=RAR-INCR; LIT=0; A5:  BARA=LIT-W(7); BUFB=R2 R4; A1:  Y1
(5)=X1(2)+Y1(2); A2:  Y2(5)=X2(2)-Y2(2); A3:  X1(5)=Y2(2)+X2(2); A4:
X2(5)=Y1(2)-X1(2); A6:  BARB=BARB-INCB; $

NLP5# IFNOT CTRJ THEN GO TO ACSAR; Z=ROM; Y(1)=RSHBUFA; W(6)=A5; A5:  BA
RA=LIT+W(6)+1; A1:  R1=P 1-P2; A2:R2=Y1(3)+A1;
 A3:  R3=P3+P4; A4:  R4=Y2(3)-A3; $

RFT31#  A1: R2=Y1(3)-R1; A3: R4=Y2(3)+R3; BUFB=R2R4; A6: BARB=
          BARB+INCB+1; A7: RAR=RAR-INCR; M1: X2(5)*Z1; M2: X1(5)*Z2;
          M3: X2(5)*Z2; M4: X1(5)*Z1$

RFT32#   LIT=SLP1; ACSAR=LIT; BUFB=R2R4; Z=ROM; A1:
          Y1(3)=X1(1)+Y1(1); A2: Y2(3)=X2(1)-Y2(1); A3: X1(3)=Y2(1)
          +X2(1); A4: X2(3)=Y1(1)-X1(1); A6: BARB=BARB-INCB$

RFT33#   LIT=-1; A5: BOTH=LIT+LSHW(1); "N/2-1 TO INCA" A1: R1=P1-P2;
          A2: R2=Y1(5)+A1; A3: R3=P3+P4; A4: R4=Y2(5)-A3;CTRJ=LSHW(1);
          M1: X2(3)*Z1; M2: X1(3)*Z2; M3: X2(3)*Z2; M4: X1(3)*Z1$

RFT34#  DECJ; BUFB=R2R4; A1: R2=Y1(5)-R1; A3: R4=Y2(5)
          +R3; A6: BARB=BARB+INCB+1; A5: BARA=BARA+W(6)$

RFT35#  BUFB=R2R4; A6: BARB=BARB-INCB; A1: R1=P1-P2; A2: R2=Y1(3)+A1;
          A3: R3=P3+P4; A4: R4=Y2(3)-A3$

RFT36#  BUFB=R2R4; A1: R2=Y1(3)-R1; A3: R4=Y2(3)+R3; A6: BARB=BARB+INCB
          +1$

RFT37#  BUFB=R2R4; W(6)=A5; LIT=0; A5: LIT+INCA+1$

RFT38#  DECJ; X(2)=RSHBUFB; A6: BARB=BARB-W(6)$

"  THE FOLLOWING LOOP SCALES THE DATA BY TWO TO KEEP SCALING THE SAME
          AS THE FFT
"
SLP1# IF CTRJ THEN SKIP; A6: BARB=BARB+INCB; BUFA=X(2); A5: BARA=BARA
          -W(6); X(1)=RSHBUFB$

SLP2#  DECJ; A6: BARB=BARB-W(6); BUFA=X(1); A5: BARA=BARA+INCA; X(2)=
          RSHBUFB; GO TO ACSAR$

RFT41#  X(2)=BUFB; A6: BARB=BARB-W(6); BUFA=X(1); A5: BARA=BARA+INCA$

RFT42#  BUFA=X(2); A5: BARA=BARA-W(6); X(1)=BUFB$

RFT43#  BUFA=X(1); INTERRUPT$
```

```
 73/03/07. 15.25.40.
PROGRAM   OCT

   "          SPAU PROGRAM  'OCT'

      W(0)=MACRO ADDRESS      W(1)=NO. OF POINTS INPUT
      W(2)=ADDRESS IN BUFB OF FEEDBACK TERMS CARRIED OVER
            (FOUR FOR OCTAVE 8; SIX FOR OTHER OCTAVES)
      W(3)=SECOND RAR OF LAST-USED FILTER
      W(4)=0              W(5)=1
      W(6)=LATEST COMPLEX FILTER OUTPUT ADDRESS (START AT 512)
      W(7)=LATEST REAL FILTER INPUT ADDRESS (BUFA)
      W(8)=LATEST REAL FILTER OUTPUT ADDRESS (BUFA)
      W(9)=4, RESET VALUE FOR MODULO COUNTER, CTRK
      W(10)=OFFSET INITIALIZATION VALUE FOR WORD COUNTER, CTRI

 NOTE:---W(2) IS NOT CURRENTLY IN USE.    "


INIT# A5:BARA=LIT-INCA; A6:BARB=LIT-INCB; LIT=1;
 A7:INCR=LIT; W(5)=A7$

I1# A5:INCA=BARA+INCA; A6:INCB=BARB+INCB; A7:LIT;
 W(6)=A7; LIT=512$

I2# A7:RAR=LIT; W(3)=A7; LIT=2048$

MAIN# A5:BARA-INCA; W(4)=A5; CALL DEMOD$

M4# CTRJ=LIT; LIT=7$

M5# DECJ; CALL RFIL$

M6# IF NOT CTRJ THEN GO TO M5$

M7# INTERRUPT$

RFIL# A5:BARA=W(4); A7:RAR=RAR+INCR; CTRK=LIT; LIT=2;
 A1:X1(15)÷Y1(15); X1(1)=A1; X2(1)=A1$
    "INCREMENT RAR TO ROM ADDRESS OF NEXT (I.E.THIS) SET OF
      FILTER COEFFICIENTS.  USE X(15) TO CLEAR FEEDBACK TERMS;
      THEY WOULD NOT BE ZERO AFTER THEIR FIRST USAGE. THIS
      WOULD NOT BE DONE IF FEEDBACK WAS BEING CARRIED OVER
      BETWEEN INPUT BATCHES."

R9# A7:RAR=RAR+INCR; Z=ROM;
 A1:X1(15)+Y1(15); X1(2)=A1; X2(2)=A1$

R10# A7:LIT+RSHW(1); W(1)=A7; CTRI=RSHW(1); Y(1)=BUFA$
    "SET CTRI AND W(1) TO NUMBER OF INPUT WORDS
      (=HALF NUMBER OF INPUT POINTS). "
```

```
R11# M1:X1(1)*Z1; M2:X2(1)*Z2; M3:X1(1)*Z3; M4:X2(1)*Z4;
 A7:LIT; W(8)=A7; Z=ROM; LIT=-1$

R12# M1:X1(2)*Z1; M2:X2(2)*Z2; M3:X1(2)*Z3; M4:X2(2)*Z4; P=BIT3;
 A7:RAR=RAR-INCR$

R13# A1:P1+P2; A2:X2(1)=Y1(1)+A1; A3:P3+P4; A4:Y1(2)=Y1(1)+A3;
 A6:BARB=LIT+W(2); A7:RAR=RAR+INCR; Z=ROM; P=BIT3$
    " X IS THE FEEDBACK TERM. Y IS THE OUTPUT OF
      THE TWO-POLE."

R14# M1:X2(1)*Z1; M2:X1(1)*Z2; M3:X2(1)*Z3; M4:X1(1)*Z4;
 A1:P1+P2; A2:X2(2)=Y1(2)+A1; A7:R7=LIT; LIT=R13; Z=ROM; DECK$
    "REVERSE ROLES OF FEEDBACK STORES, X1 AND X2.
      DO NOT GENERATE FIRST OUTPUT POINT IN A4."

R15# M1:X2(2)*Z1; M2:X1(2)*Z2; M3:X2(2)*Z3; M4:X1(2)*Z4; P=BIT3;
 A7:RAR=RAR-INCR; DECI$

R16# A1:P1+P2; A2:X1(1)=Y2(1)+A1; A3:P3+P4; A4:Y2(2)=Y2(1)+A3;
 Z=ROM; A5:BARA=BARA+INCA; W(7)=A5; A7:RAR=RAR+INCR; IF CTRK THEN GO
 TO R18; P=BIT3$
    " CTRK COUNTS ALTERNATE PAIRS OF POINTS TO
      CONTROL PACKING, NOT DECIMATION."

R17# M1:X1(1)*Z1; M2:X2(1)*Z2; M3:X1(1)*Z3; M4:X2(1)*Z4; Z=ROM;
 A1:P1+P2; A2:X1(2)=Y2(2)+A1; A3:P3+P4; A4:R3=Y2(2)+A3;
 Y(1)=BUFA; GO TO R12$

R18# M1:X1(1)*Z1; M2:X2(1)*Z2; M3:X1(1)*Z3; M4:X2(1)*Z4; Z=ROM;
 A1:P1+P2; A2:X1(2)=Y2(2)+A1; A3:P3+P4; A4:R4=Y2(2)+A3;
 Y(1)=BUFA; A5:BARA=LIT+W(8)+1; W(8)=A5$

R19# M1:X1(2)*Z1; M2:X2(2)*Z2; M3:X1(2)*Z3; M4:X2(2)*Z4;
 A5:BARA=W(7); A6:BARB=BARB-INCB; A7:RAR=RAR-INCR;
 BUFA=R3R4; BUFB=X(2); P=BIT3; CTRK=LIT; LIT=2;
 IF NOT CTRI THEN GO TO R7$
    "STORE FEEDBACK TERMS IN BUFB FOR POSSIBLE
      CARRYOVER TO THE NEXT BATCH."

R20# A7:RAR=RAR+INCR; BUFB=X(1); W(3)=A7$

DEMOD# A5:BARA=W(4); A6:BARB=W(4); A7:RAR=LIT; LIT=2079$
    "FIRST TWO OSCILLATOR PHASES ARE IN ROM(2079)."

D22# A5:BARA=BARA+INCA; A6:INCB=LIT+RSHW(1); LIT=-3;
 A7:RAR=RAR+INCR; Z=ROM; X(0)=BUFA$
```

```
D23# M1:X1(Ø)*Z1; M2:X1(Ø)*Z2; M3:X2(Ø)*Z3; M4:X2(Ø)*Z4;
 A5:BARA=BARA+INCA; A7:LIT; LIT=4; W(9)=A7; Z=ROM;
 Y(Ø)=BUFA; CTRK=LIT$
    "DEMODULATE TWO INPUT POINTS (IN X1 AND X2) AT
      A TIME."

D24# A6:LIT+INCB; W(1Ø)=A6; DECK$

D25# M1:Y1(Ø)*Z1; M2:Y1(Ø)*Z2; M3:Y2(Ø)*Z3; M4:Y2(Ø)*Z4;
 A6:INCB=W(5); A7:RAR=RAR+INCR; X(Ø)=BUFA; DECK;
 R1=P1; R2=P2; R3=P3; R4=P4$
    " CTRI COUNTS INPUT WORDS."

D26# A5:BARA=BARA+INCA; A6:BARB=BARB+INCB; A7:R7=LIT;
 LIT=D28; Z=ROM; CTRI=W(1Ø); BUFB=R1R2$
    "OUTPUT THE FIRST POINT AS A QUAD PAIR. OFFSET
      CTRI BY 3 TO ACCOUNT FOR MISSED READS."

D27# M1:X1(Ø)*Z1; M2:X1(Ø)*Z2; M3:X2(Ø)*Z3; M4:X2(Ø)*Z4;
 A6:BARB=BARB+INCB; A7:RAR=RAR+INCR; Y(Ø)=BUFA;
 R1=P1; R2=P2; R3=P3; R4=P4; BUFB=R3R4; DECK$
    "OUTPUT SECOND POINT AS A QUAD PAIR."

D28# A5:BARA=BARA+INCA; A6:BARB=BARB+INCB;
 Z=ROM; BUFB=R1R2; DECI; IF CTRK THEN GO TO D32$
    "OUTPUT ONE QUAD PAIR TO BUFB ON EVERY CLOCK."

D29# M1:Y1(Ø)*Z1; M2:Y1(Ø)*Z2; M3:Y2(Ø)*Z3; M4:Y2(Ø)*Z4;
 A6:BARB=BARB+INCB; A7:RAR=RAR+INCR; X(Ø)=BUFA; BUFB=R3R4;
 R1=P1; R2=P2; R3=P3; R4=P4; DECK; IF CTRI THEN GO TO D34$

D3Ø# A5:BARA=BARA+INCA; A6:BARB=BARB+INCB; Z=ROM; BUFB=R1R2;
 DECI; IF NOT CTRK THEN GO TO D27$
    " CTRK COUNTS MODULO 5 (5 ROM WORDS = 1Ø PHASE ANGLES)."

D31# M1:X1(Ø)*Z1; M2:X1(Ø)*Z2; M3:X2(Ø)*Z3; M4:X2(Ø)*Z4;
 A6:BARB=BARB+INCB; A7:RAR=LIT; LIT=2Ø79; BUFB=R3R4;
 CTRK=W(9); R1=P1; R2=P2; R3=P3; R4=P4; Y(Ø)=BUFA; GO TO R7$

D32# M1:Y1(Ø)*Z1; M2:Y1(Ø)*Z2; M3:Y2(Ø)*Z3; M4:Y2(Ø)*Z4;
 A6:BARB=BARB+INCB; A7:RAR=RAR-W(9); X(Ø)=BUFA; BUFB=R3R4;
 R1=P1; R2=P2; R3=P3; R4=P4; CTRK=W(9); IF CTRI THEN GO TO D34$

D33# A5:BARA=BARA+INCA; A6:BARB=BARB+INCB; Z=ROM; BUFB=R1R2;
 DECI; GO TO D27$

D34# A6:BARB=BARB+INCB; BUFB=R1R2$

D35# A6:BARB=BARB+INCB; BUFB=R3R4; R1=P1; R2=P2; R3=P3;
 R4=P4$
```

```
CFIL# A6:BARB=BARB+INCB; A7:RAR=LIT+W(3); LIT=1;
 W(3)=A7; BUFB=R1R2$
    "INITIALIZE RAR TO 1 + ADDRESS OF LAST=USED FILTER."

C37# A6:BARB=W(4); A7:RAR=RAR+INCR; Z=ROM; BUFB=R3R4;
 CTRK=LIT; LIT=2$
    "FINAL OUTPUT FROM DEMOD TO BUFB."

C38# M1:X1(1)*Z1; M2:X2(1)*Z2; M3:X1(1)*Z3; M4:X2(1)*Z4;
 A5:BARA=W(6); Y(1)=BUFB; A7:RAR=RAR-INCR; Z=ROM$
    "MULTIPLY FOR FIRST TWO-POLE OF SINE CHANNEL."

C39# M1:X1(2)*Z1; M2:X2(2)*Z2; M3:X1(2)*Z3; M4:X2(2)*Z4;
 A7:RAR=RAR+INCR; Z=ROM; CTRI=RSHW(1); P=BIT3$
    "SET CTRI (BUT NOT W(1)) TO HALF NUMBER OF INPUT
      WORDS.  SECOND TWO-POLE OF SINE CHANNEL."

C40# M1:X1(3)*Z1; M2:X2(3)*Z2; M3:X1(3)*Z3; M4:X2(3)*Z4;
 A1:P1+P2; A2:X2(1)=Y1(1)+A1; A3:P3+P4; A4:Y1(2)=Y1(1)+A3;
 A7:RAR=RAR-INCR; Z=ROM; P=BIT3$
    "FIRST TWO-POLE OF COSINE CHANNEL."

C41# M1:X1(4)*Z1; M2:X2(4)*Z2; M3:X1(4)*Z3; M4:X2(4)*Z4;
 A1:P1+P2; A2:X2(2)=Y1(2)+A1; A7:RAR=RAR+INCR; Z=ROM;
 DECK; P=BIT3$
    " CTRK COUNTS FOR 2:1 COUNTDOWN OF GENEREATED
      PAIRS. DO NOT GENERATE FIRST SINE OUTPUT IN
      A4.  SECOND TWO-POLE OF COSINE CHANNEL."

C42# M1:X2(1)*Z1; M2:X1(1)*Z2; M3:X2(1)*Z3; M4:X1(1)*Z4;
 A1:P1+P2; A2:X2(3)=Y2(1)+A1; A3:P3+P4; A4:Y1(2)=Y2(1)+A3;
 Z=ROM; A6:BARB=BARB+INCB; A7:RAR=RAR-INCR; P=BIT3$
    "REVERSE ROLES OF FEEDBACK STORES X1 AND X2."

C43# M1:X2(2)*Z1; M2:X1(2)*Z2; M3:X2(2)*Z3; M4:X1(2)*Z4;
 A1:P1+P2; A2:X2(4)=Y1(2)+A1; A7:RAR=RAR+INCR; Z=ROM;
 Y(1)=BUFB; P=BIT3$
    "DO NOT GENERATE FIRST COSINE OUTPUT IN A4."

C44# M1:X2(3)*Z1; M2:X1(3)*Z2; M3:X2(3)*Z3; M4:X1(3)*Z4;
 A1:P1+P2; A2:X1(1)=Y1(1)+A1; A3:P3+P4; A4:Y1(2)=Y1(1)+A3;
 A7:RAR=RAR-INCR; Z=ROM; DECI; P=BIT3$
    " CTRI COUNTS ONCE FOR EVERY TWO READS OF BUFB
      (AT C43 AND C47)."

C45# M1:X2(4)*Z1; M2:X1(4)*Z2; M3:X2(4)*Z3; M4:X1(4)*Z4;
 A1:P1+P2; A2:X1(2)=Y1(2)+A1; A3:P3+P4; A4:R3=Y1(2)+A3;
 A7:RAR=RAR+INCR; Z=ROM; P=BIT3$
    "PACK SINE OUTPUT IN R3."
```

```
C46# M1:X1(1)*Z1; M2:X2(1)*Z2; M3:X1(1)*Z3; M4:X2(1)*Z4;
 A1:P1+P2; A2:X1(3)=Y2(1)+A1; A3:P3+P4; A4:Y1(2)=Y2(1)+A3;
 A7:RAR=RAR-INCR; A6:BARB=BARB+INCB; Z=ROM; P=BIT3$

C47# M1:X1(2)*Z1; M2:X2(2)*Z2; M3:X1(2)*Z3; M4:X2(2)*Z4;
 A1:P1+P2; A2:X1(4)=Y1(2)+A1; A3:P3+P4; A4:R4=Y1(2)+A3;
 A7:RAR=RAR+INCR; Z=ROM; Y(1)=BUFB; IF NOT CTRK THEN GO
 TO C40; P=BIT3$
    "PACK COSINE OUTPUT IN R4. IF CTRK IS NOT ZERO,
      DISCARD R3R4."

C48# M1:X1(3)*Z1; M2:X2(3)*Z2; M3:X1(3)*Z3; M4:X2(3)*Z4;
 A1:P1+P2; A2:X2(1)=Y1(1)+A1; A3:P3+P4; A4:Y1(2)=Y1(1)+A3;
 A5:BARA=BARA+INCA; A7:RAR=RAR-INCR; W(6)=A5; BUFA=R3R4;
 Z=ROM; CTRK=LIT; LIT=2; IF CTRI THEN GO TO ACSAR; P=BIT3$
 .
C49# M1:X1(4)*Z1; M2:X2(4)*Z2; M3:X1(4)*Z3; M4:X2(4)*Z4;
 A1:P1+P2; A2:X2(2)=Y1(2)+A1; A7:RAR=RAR+INCR; Z=ROM;
 DECK; GO TO C42; P=BIT3$
```

```
    WGTSUM:
          THIS PROGRAM GIVES A LOCAL AVERAGE OF FOUR ELEMENTS WHOSE
          INITIAL ADDRESS IS SPECIFIED IN AN INDIRECT ADDRESS(IA)
          MODE

          W(3) CONTAINS ADDRESS OF IA TABLE IN BUFA.
          W(4) CONTAINS ADDRESS OF COEFFICIENT TABLE IN BUFB.
          W(2) CONTAINS NUMBER OF POINTS IN IA TABLE

          REAL PART OF IA TABLE WORD CONTAINS IA.
         OUTPUT IS WRITTEN INTO CORRESPONDING IA TABLE ADDRESS
"
WS0#  CTRJ=W(2); A6: BOTH=INCB$"LOAD NO. OF PTS. TO CTR"

WS1#  A5: BOTH=W(3); A6: INCB=BARB-INCB$"ZERO INCB, LOAD IA TABLE
          ADDRESS"

WS2#  X(6)=BUFA; DECJ; A5: BOTH=LIT+INCA+1; A6: BARB=W(4); LIT=0$
          "LOAD COEFF. TABLE ADDRESS"


WS3#  X(5)=BUFA; DECJ; W(5)=X1(6);"TRANSFER IA TO ADDRESS SECTION"
          Y(1)=BUFB; A6: BARB=BARB+INCB+1; ACSAR=LIT; LIT=WS14$

WS4#  LIT=0; Y(2)=BUFB; A5: BARA=LIT+W(5); A6: BARB=BARB+INCB+1$

WS5#  X(1)=BUFA; W(5)=A5; LIT=1; A5: BARA=LIT+W(5)$

WS6#  X(2)=BUFA; W(5)=A5; LIT=1; A5: BARA=LIT+W(5); M1: X1(1)*Y1(1);
          M2: X2(1)*Y2(1); M3: X2(1)*Y1(1); M4: X1(1)*Y2(1)$

WS7#  M1: X1(2)*Y1(2); M2: X2(2)*Y2(2); M3: X2(2)*Y1(2);
          M4: X1(2)*Y2(2); GO TO L4$

L0#   IF CTRJ THEN GO TO ACSAR; " EXIT BEFORE N+1 ST X(5)" X(1)=BUFA;
          LIT=1; W(5)=A5; M1: X1(4)*Y1(4); M2: X2(4)*Y2(4); M3:
          X2(4)*Y1(4); M4: X1(4)*Y2(4); A5: BARA=LIT+W(5); A6: BARB=
          BARB+INCB+1$

L1#   Y(1)=BUFB; A6: BARB=BARB+INCB+1; X(2)=BUFA; A5: BOTH=LIT+INCA+1;
          LIT=0; A1: P1-P2; A2: R2=R2+A1; A3: P3+P4; A4: R4=
          R4+A3$

L2#   X(5)=BUFA;"GET NEW IAW" DECJ; Y(2)=BUFB; M1: X1(1)*Y1(1);
          M2: X2(1)*Y2(1); M3: X2(1)*Y1(1); M4: X1(1)*Y2(1);
          A1: P1-P2; A2: R2=R2+A1; A3: P3+P4; A4: R4=R4+A3;
          A5: BARA=LIT+INCA; LIT=-2$"MUST DECREASE ADDRESS TO
          ACCOUNT FOR DELAY IN GETTING RESULT"

L3#   BUFA=R2R4;"OUTPUT RESULT" LIT=1; W(5)=A5; A5: BARA=LIT+W(5);
          A6: BARB=BARB+INCB+1; M1: X1(2)*Y1(2); M2: X2(2)*Y2(2);
          M3: X2(2)*Y1(2); M4: X1(2)*Y2(2)$
```

```
L4#   "ENTER LOOP" X(3)=BUFA; A5: BARA=LIT+W(5); LIT=1; Y(3)=BUFB;
          A6: BARB=BARB+INCB+1; W(5)=X1(5);"TRANSFER IAW" A1:
          R2=P1-P2; A3: R4=P3+P4$

L5#   X(4)=BUFA; Y(4)=BUFB; GO TO L0; A5: BARA=W(5); A1: P1-P2;
          A2: R2=R2+A1; A3: P3+P4; A4: R4=R4+A3; M1: X1(3)*Y1(3);
          M2: X2(3)*Y2(3); M3: X2(3)*Y1(3); M4: X1(3)*Y2(3)$


WS14#   A6: BARB=BARB+INCB+1; Y(1)=BUFB;; A5: BARA=LIT+W(5); W(5)=A5;
          X(2)=BUFA; A1: P1-P2; A2: R2=R2+A1; A3: P3+P4; A4: R4=R4+A3;
          LIT=1$

WS15#   X(3)=BUFA; A5: BARA=LIT+INCA; Y(2)=BUFB; A6: BARB=BARB+INCB+1;
          M1: X1(1)*Y1(1); M2: X2(1)*Y2(1); M3: X2(1)*Y1(1);
          M4: X1(1)*Y2(1); A1: P1-P2; A2: R2=R2+A1; A3: P3+P4;
          A4: R4=R4+A3; LIT=-1$

WS16#   BUFA=R2R4; A5: BARA=LIT+W(5); LIT=1; W(5)=A5; Y(3)=BUFB;
          A6: BARB=BARB+INCB+1; M1: X1(2)*Y1(2); M2: X2(2)*Y2(2);
          M3: X2(2)*Y1(2); M4: X1(2)*Y2(2)$

WS17#   X(4)=BUFA; A1: R2=P1-P2; A3: R4=P3+P4; M1: X1(3)*Y1(3);
          M2: X2(3)*Y2(3); M3: X2(3)*Y1(3); M4: X1(3)*Y2(3);
          Y(4)=BUFB$

WS18#   A1: P1-P2; A2: R2=R2+A1; A3: P3+P4; A4: R4=R4+A3; M1: X1(4)*
          Y1(4); M2: X2(4)*Y2(4); M3: X2(4)*Y1(4); M4:X1(4)*Y2(4)$

WS19#   A1: P1-P2; A2: R2=R2+A1; A3: P3+P4; A4: R4=R4+A3$

WS20#   A1: P1-P2; A2: R2=R2+A1; A3: P3+P4; A4: R4=R4+A3;
          A5: BARA=INCA$

WS21#   BUFA=R2R4; INTERRUPT$
```

```
"                              XCORREL

   THIS CROSS-CORRELATION PROGRAM IS A LINKING OF THREE MACROS: CONJG,
        MLT,LFFT.  THE ADDRESS OF LFFT MUST BE IN W(0); THE ADDRESS
        OF CONJG MUST BE IN W(11).  THE EXIT ADDRESS MUST BE IN W(10)
        HERE THE EXIT ADDRESS, DONE, IS AN INTERRUPT.  W(3) AND W(4)
        CONTAIN THE STARTING ADDRESS OF THE INPUT ARRAYS,ONE IN BUFA
        ONE IN BUFB.  OTHER PARAMETERS IN W-STORE ARE DEFINED IN THE
        MACROS THAT UTILIZE THEM.     "

"  MLT: VECTOR IN BUFA IS MULTIPLIED BY VECTOR IN BUFB, RESULT AND
        SOURCE VECTORS START IN LOCATION IN W(3)"

MLT0# LIT=0; A7: LIT; W(7)=A7$

MLT1# LIT=-2; W(8)=A7; A7: LIT$

MLT2# A5: BOTH=W(3); A6: BOTH=W(3)$"LOAD START ADDRESSES"

MLT3# CTRJ=LSHW(1); A5: INCA=BARA-INCA; A6: INCB=BARB-INCB$"ZERO
        INC REGISTERS, N TO CTRJ"

MLT4# DECJ; ACSAR=LIT; LIT=LL; X(1)=BUFA; Y(1)=BUFB; A5: BARA=BARA+
        INCA+1; A6: BARB=BARB+INCB+1; W(7)=A6$"SET UP LOOP ON LL,
        SAVE BARB IN W(7) FOR DELAYED OUTPUT OF RESULT"

MLT5# DECJ; M1: X1(1)*Y1(1); M2: X2(1)*Y2(1); M3: X2(1)*Y1(1); M4:
        X1(1)*Y2(1); Y(2)=BUFB; A6: BARB=BARB+INCB+1; W(7)=A6;
        LIT=MLT11; A7: R7=LIT$"SET LOOP ESCAPE"

MLT6# DECJ; X(2)=BUFA; A5: BARA=BARA+INCA+1$


LL# Y(1)=BUFB; A6: BARB=BARB+ W(8); A1: R2=P1-P2; A3: R4=P3+P4; M1:
      X1(2)*Y1(2); M2: X2(2)*Y2(2); M3: X2(2)*Y1(2); M4: X1(2)*
      Y2(2)$

LM# DECJ; X(1)=BUFA; A5: BARA=BARA+INCA+1; BUFB=R2R4; A6: BARB=LIT+W(7);
 LIT=1
; W(7)=A6$

LN# Y(2)=BUFB; M1: X1(1)*Y1(1); M2: X2(1)*Y2(1); M3: X2(1)*Y1(1)
     ; M4: X1(1)*Y2(1); A1: R2=P1-P2; A3: R4=P3+P4; A6: BARB=BARB+W(8);
 IF CTRJ THEN GO TO R7$
LO# DECJ; X(2)=BUFA; BUFB=R2R4; A6: BARB=LIT+W(7) ; LIT=1; W(7)=A6; A5:
BARA=B
ARA+INCA+1;GO TO ACSARS

MLT11# X(2)=BUFA; BUFB=R2R4; A6: BARB=BARB+INCB+1$"EMPTY PIPE"

MLT12# M1: X1(2)*Y1(2); M2: X2(2)*Y2(2); M3: X2(2)*Y1(2);
        M4: X1(2)*Y2(2); A1: R2=P1-P2; A3: R4=P3+P4$

MLT13# BUFB=R2R4; A6: BARB=BARB+INCB+1$

MLT14# LIT=0; A7: R7=LIT+W(0); A1: R2=P1-P2; A3: R4=P3+P4$

MLT15# BUFB=R2R4; SWAP; GO TO R7$"RESULT TO BUFA, EXIT TO ADDRESS IN
        W(0)"
```

```
"     CONJG:  CONJUGATES DATA AND SHIFTS BY AMOUNT IN W(12)  "

CONJG#  A5: BOTH=W(3); A6: BOTH=W(3)$

CONJG1# LIT=0; A7: R7=LIT+W(0); Y1(1)=LIT; A5: INCA=BARA-INCA; A6: INCB
          =BARB-INCB $ "SET EXIT ADDRESS IN R7, ZERO INC REGISTERS, LOAD
          STARTING ADDRESS FROM W(3)"

CONJG15# A6: BARB=BARB+W(12)$

CONJG2# CTRJ=W(1); X(1)=BUFA; A5: BARA=BARA+INCA+1; A7: RAR=LIT+W(1);
          LIT=-1$
          "SET CTRJ AT N/2, PRPARE CTRI WITH DECREMENT"

CONJG3# DECJ; X(2)=BUFA; A5: BARA=BARA+INCA+1; A1: R2=X1(1)+Y1(1);
          A3: R4=Y1(1)-X2(1); A7: RAR-RSHW(12); W(12)=A7; LIT=LP1;
          ACSAR=LIT$  "LOOP ON LP1, DECJ FOR X(1) READ, CONJG. FST.
          DATA"

CONJG4# CTRI=W(12); LIT=0; A7: LIT; W(12)=A7; " SET CTRI =(N-W(12))/2-1,
          ZERO W(12) SO NO SHIFT ON NEXT CALL"$

LP1#   DECJ; BUFB=R2R4; X(1)=BUFA; A1: R2=X1(2)+Y1(1); A3: R4=
     Y1(1)-X2(2); A5: BARA=BARA+INCA+1; A6: BARB=BARB+INCB+1;
     IF CTRI THEN GO TO ROT   "LOOP ON LP1, ROT COMPLETES THE CIRCULAR
     SHIFT OF THE DATA POINTS"  $

LP2# DECI; IF NOT CTRJ THEN GO TO ACSAR; A1: R2=X1(1)+Y1(1); A3: R4=
      Y1(1)-X2(1); A6: BARB=BARB+INCB+1; BUFB=R2R4; X(2)=BUFA;
      A5: BARA=BARA+INCA+1$

CONJG7# BUFB=R2R4; A1: R2=X1(2)+Y1(1); A3: R4=Y1(1)-X2(2);
     A5: INCA=W(10); A6: BARB=BARB+INCB+1$

CONJG8# BUFB=R2R4;SWAP; A5: INCA; W(0)=A5;
      " HERE WE SET ACSAR FOR FFT RETURN, MAKE W(10) CONTENTS NEW
        CONJG EXIT"  : GO TO R7$

ROT#  CTRI=W(1)   ; A1: R2=X1(1)+Y1(1); A3: R4=Y1(1)-X2(1);
      A5: BARA=BARA+INCA+1; BUFB=R2R4; X(2)=BUFA; A6: BARB=INCB;
      IF NOT CTRJ THEN GO TO ACSAR$

ROT1# GO TO CONJG7$"THIS RETURN NEEEDED FOR CASE W(12)=2"

DONE# INTERRUPTS
     "   SPAU PROGRAM    'LFFT'

   W(1)=N/2  W(2)=LOG2(N)  W(3)= DATA ADDRESS  W(4)= RESULT ADDRESS
   W(5)&CTRK=BUTTERFLIES PER HALF GROUP
   W(6)&CTRJ=GROUPS PER STAGE
     CTRI=STAGE      W(8)=ROM INCREMENT  W(9)=1  "


L0# A6:BARB=INCB; A7:RAR=LIT+INCR; A5:INCA=W(1); CTRJ=W(1)$

L1# A5:BARA=W(3); A6:INCB=BARB-INCB; A7:RAR=RAR-INCR$

L2# A5:BARA=BARA+INCA; A6:INCB=LIT+INCB; LIT=1; CTRK=LIT; W(5)=A6$
```

```
L3# A5:BARA=BARA-INCA; X(1)=RSHBUFA; Z=ROM$

L4# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:INCB=LIT-INCB; Y(1)=RSHBUFA;
  W(6)=CTRJ$

L5# M1:X2(1)*Z1; A5:BARA=BARA-INCA; X(2)=RSHBUFA; CTRI=W(2); DECJ$

L6# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=W(4); Y(2)=RSHBUFA; DECJ;
  A1:R1=P1+P2; A2:R2=Y1(1)+A1; A3:R3=P3-P4; A4:R4=Y2(1)+A3$

L7# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; A7:LIT;
  LIT=1; X(1)=RWHBUFA; BUFB=R2R4; A1:R1=Y1(1)-R1; A3:R3=Y2(1)-R3;
  DECJ; W(9)=A7$

L8# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB-INCB; Y(1)=RSHBUFA;
  BUFB=R1R3; A1:R1=P1+P2; A2:R2=Y1(2)+A1; A3:R3=P3-P4;
  A4:R4=Y2(2)+A3$

L9# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; X(2)=RSHBUFA;
  BUFB=R2R4;A1:R1=Y1(2)-R1; A3:R3=Y2(2)-R3; DECJ; A7:INCR=LIT;
  LIT=-1024; W(8)=A7$

L10# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB-INCB; Y(2)=RSHBUFA;
  BUFB=R1R3; A1:R1=P1+P2; A2:R2=Y1(1)+A1; A3:R3=P3-P4;
  A4:R4=Y2(1)+A3; IF NOT CTRJ THEN GO TO L7$

L11# M1:X2(2)*Z1; A5:BARA=W(4); A6:BARB=BARB-INCB; BUFB=R2R4;
  A1:R1=Y1(1)-R1; A3:R3=Y2(1)-R3$

L12# A5:BARA=BARA+INCA; A6:BARB=BARB-INCB; A7:LIT+RSHW(6); BUFB=R1R3;
  A1:R1=P1+P2; A2:R2=Y1(2)+A1; A3:R3=P3-P4; A4:R4=Y2(2)+A3; W(6)=A7;
  CTRJ=RSHW(6)$

L13# A6:BARB=BARB-INCB; BUFB=R2R4; A1:R1=Y1(2)-R1; A3:R3=Y2(2)-R3;
  DECI$

L14# BUFB=R1R3; SWAP$

L15# A5:BARA=BARA-INCA; A6:INCB=LIT-LSHW(5); X(1)=RSHBUFA; Z=ROM$

L16# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A7:RAR=RAR-INCR;
  Y(1)=RSHBUFAS

L17# M1:X2(1)*Z1; A5:BARA=BARA-INCA; X(2)=RSHBUFA;
  Z=ROM$

L18# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=W(4);
  A7:RAR=RAR+INCR; Y(2)=RSHBUFA; A1:R1=P1+P2; A3:R3=P3-P4;
  A2:R2=Y1(1)+A1; A4:R4=Y2(1)+A3; DECJ$

L19# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; X(1)=RSHBUFA;
  BUFB=R2R4; Z=ROM; A1:R1=Y1(1)-R1; A3:R3=Y2(1)-R3$

L20# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
  A7:RAR=RAR-INCR; Y(1)=RSHBUFA; BUFB=R1R3; A1:R1=P1+P2; A2:R2=Y1(2)+A1;
  A3:R3=P3-P4; A4:R4=Y2(2)+A3$

L21# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; X(2)=RSHBUFA;
BUFB=R2R4; A1:R1=Y1(2)-R1; A3:R3=Y2(2)-R3;DECJ; Z=ROM$
```

```
L22# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+W(9); Y(2)=RSHBUFA;
 A7:RAR=RAR+INCR; BUFB=R1R3; A1:R1=P1+P2; A2:R2=Y1(1)+A1;
 A3:R3=P3-P4; A4:R4=Y2(1)+A3; IF NOT CTRJ THEN GO TO L19$

L23# M1:X2(2)*Z1; A5:BARA=W(4); A6:BARB=BARB-INCB; BUFB=R2R4;
 A1:R1=Y1(1)-R1; A3:R3=Y2(1)-R3$

L24# A6:BARB=BARB+INCB+1; A5:LIT+RSHW(6); CTRH=RSHW(6); W(6)=A5;
 BUFB=R1R3; A1:R1=P1+P2; A2:R2=Y1(2)+A1; A3:R3=P3-P4; A4:R4=Y2(2)+A3$

L25# A6:BARB=BARB-INCB;
 BUFB=R2R4; DECI; A1:B1=Y1(2)-R1; A3:R3=Y2(2)-R3$

L26# A5:BARA=BARA+INCA; A6:BARB=LIT+INCB; W(5)=A7; BUFB=R1R3;
 CTRK=LSHW(5); SWAP; A7:LIT+LSHW(5)$

L27# A5:BARA=BARA-INCA; A6:INCB=BARB+INCB; X(1)=RSHBUFA; Z=ROM;
 A7:INCR=LIT+RSHW(8); W(8)=A7$

L28# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=LIT+W(4);
 A7:RAR=RAR-INCR; Y(1)=RSHBUFA; LIT=-1; DECK$

L29# M1:X2(1)*Z1; A5:BARA=BARA-INCA; X(2)=RSHBUFA; Z=ROM; DECK$

L30# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A7:RAR=RAR-INCR; Y(2)=RSHBUFA;
 A1:R1=P1+P2; A2:R2=Y1(1)+A1; A3:R3=P3-P4; A4:R4=Y2(1)+A3; IF CTRK
 THEN GO TO L51; A6:BARB=BARB+W(9)$

L31# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB;
 X(1)=RWHBUFA; BUFB=R2R4; Z=ROM; A1:R1=Y1(1)-R1; A3:R3=Y2(1)-R3$

L32# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR-INCR; Y(1)=RSHBUFA; BUFB=R1R3; DECK; A1:R1=P1+P2;
 A2:R2=Y1(2)+A1; A3:R3=P3-P4; A4:R4=Y2(2)+A3$

L33# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB;
 X(2)=RSHBUFA; BUFB=R2R4; DECK; Z=ROM; A1:R1=Y1(2)-R1;
 A3:R3=Y2(2)-R3$

L34# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR-INCR; Y(2)=RSHBUFA; BUFB=R1R3; A1:R1=P1+P2;
 A2:R2=Y1(1)+A1; A3:R3=P3-P4; A4:R4=Y2(1)+A3; IF NOT CTRK
 THEN GO TO L31$

L35# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB;
 X(1)=RSHBUFA; BUFB=R2R4; CTRK=W(5); Z=ROM; A1:R1=Y1(1)-R1;
 A3:R3=Y2(1)-R3$

L76# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1;
 A6:BARB=BARB+INCB+1; A7:RAR=RAR+INCR; Y(1)=RSHBUFA;
 BUFB=R1R3; DECK; A1:R1=P1+P2; A2:R2=Y1(2)+A1; A3:R3=P3-P4;
 A4:R4=Y2(2)+A3$

L37# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB;
 X(2)=RSHBUFA; BUFB=R2R4; DECK; Z=ROM; A1:R1=Y1(2)-R1;
 A3:R3=Y2(2)-R3$

L38# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1;
 A6:BARB=BARB+INCB+1; A7:RAR=RAR+INCR; Y(2)=RSHBUFA; BUFB=R1R3;
 A1:R1=P1-P2; A2:R2=Y1(1)+A1; A3:R3=P3+P4; A4:R4=Y2(1)-A3$
```

```
L39# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; DECK;
 X(1)=RSHBUFA; BUFB=R2R4; Z=ROM; A1:R1=Y1(1)-R1; A3:R3=Y2(1)+R3$

L40# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR+INCR; Y(1)=RSHBUFA; BUFB=R1R3;
 A1:R1=P1-P2; A2:R2=Y1(2)+A1; A3:R3=P3+P4; A4:R4=Y2(2)-A3$

L41# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB;
 X(2)=RSHBUFA; BUFB=R2R4; DECK; Z=ROM; A1:R1=Y1(2)-R1;
 A3:R3=Y2(2)+R3$

L42# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 BUFB=R1R3; A1:R1=P1-P2; A2:R2=Y1(1)+A1; A3:R3=P3+P4;
 A4:R4=Y2(1)-A3; A7:RAR=RAR+INCR; IF NOT CTRK THEN GO TO L39;
 Y(2)=RSHBUFA$

L43# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; Z=ROM;
 X(1)=RSHBUFA; BUFB=R2R4; CTRK=W(5); DECJ; A1:R1=Y1(1)-R1;
 A3:R3=Y2(1)+R3$

L44# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR-INCR; Y(1)=RSHBUFA; BUFB=R1R3; A1:R1=P1-P2;
 A2:R2=Y1(2)+A1; A3:R3=P3+P4; A4:R4=Y2(2)-A3; DECK$

L45# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; Z=ROM;
 BUFB=R2R4; A1:R1=Y1(2)-R1; A3:R3=Y2(2)+R3; DECK; X(2)=RSHBUFA;
 ACSAR=LIT; LIT=L60$

L46# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; Y(2)=RSHBUFA; A7:RAR=RAR-INCR;
 A1:R1=P1+P2; A2:R2=Y1(1)+A1; A3:R3=P3-P4; A4:R4=Y2(1)+A3;BUFB=R1R3;
 A6:BARB=BARB+W(9); IF NOT CTRJ THEN GO TO L31$

L47# A5:BARA=W(4); A7:RAR=LIT; DECI; SWAP$

L48# A5:BARA=BARA+INCA; A6:RSHW(6); W(6)=A6; CTRJ=RSHW(6); IF
 CTRI THEN GO TO ACSAR; A7:R7=LIT; LIT=L27$

L50# A5:LSHW(5); W(5)=A5; CTRK=LSHW(5); GO TO R7; A6:BARB=LIT+INCB$

L51# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; Z=ROM;
 X(1)=RSHBUFA; BUFB=R2R4; A1:R1=Y1(1)-R1; A3:R3=Y2(1)-R3$

L52# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR+INCR;
 Y(1)=RSHBUFA; BUFB=R1R3; A1:R1=P1+P2; A2:R2=Y1(2)+A1;
 A3:R3=P3-P4; A4:R4=Y2(2)+A3$

L53# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; Z=ROM;
 X(2)=RSHBUFA; BUFB=R2R4; A1:R1=Y1(2)-R1; A3:R3=Y2(2)-R3$

L54# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR+INCR; Y(2)=RSHBUFA; BUFB=R1R3; A1:R1=P1-P2;
 A2:R2=Y1(1)+A1; A3:R3=P3+P4; A4:R4=Y2(1)-A3; DECJ$

L55# M1:X2(2)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; Z=ROM;
 X(1)=RSHBUFA; BUFB=R2R4; A1:R1=Y1(1)-R1; A3:R3=Y2(1)+R3$
```

```
L56# M1:X2(1)*Z1; A5:BARA=BARA+INCA+1; A6:BARB=BARB+INCB+1;
 A7:RAR=RAR-INCR; Y(1)=RSHBUFA; BUFB=R1R3; A1:R1=P1-P2;
 A2:R2=Y1(2)+A1; A3:R3=P3+P4; A4:R4=Y2(2)-A3; ACSAR=LIT; LIT=L48$

L57# M1:X2(1)*Z1; A5:BARA=BARA-INCA; A6:BARB=BARB-INCB; Z=ROM;
 X(2)=RSHBUFA; BUFB=R2R4; A1:R1=Y1(2)-R1; A3:R3=Y2(2)+R3$

L58# M1:X2(2)*Z1; A5:BARA=BARA+INCA+1; A7:RAR=RAR-INCR; Y(2)=RSHBUFA;
 A1:R1=P1+P2; A2:R2=Y1(1)+A1; A3:R3=P3-P4; A4:R4=Y2(1)+A3; BUFB=R1R3;
 A6:BARB=BARB+W(9); IF NOT CTRJ THEN GO TO L51$

L59# A5:BARA=W(4); A7:RAR=LIT; DECI;
 SWAP; GO TO ACSAR$

L60# A7: R7=LIT+W(11); LIT=0$
L61# GO TO R7 $
```

Appendix C

FAST FOURIER TRANSFORM ALGORITHM

A DFT of a sequence of N (generally) complex input points $X(j)$ is given by the equation

$$A(n) = \sum_{j=0}^{N-1} X(j) \exp \frac{2\pi ijn}{N} \quad n = 0, 1, ..., N-1 \tag{C1}$$

where i represents the square root of minus one ($i=\sqrt{-1}$). Direct evaluation of Eq. (C1) requires approximately N^2 multiplications, whereas the number for indirect FFT methods is approximately $2N \log_2 N$ when a radix 2 factorization of the equation is used.† An FFT algorithm can be derived conveniently in matrix form as follows. Consider the input and output data arrays to be complex column vectors $\mathbf{X}$ and $\mathbf{A}$, respectively, and express the DFT as a matrix product:

$$\mathbf{A} = \mathbf{W}_N \mathbf{X} \tag{C2}$$

where $\mathbf{W}_N$ is an $N \times N$ matrix whose elements are

$$\mathbf{W}_N(n,j) = \left[\exp \frac{2\pi i}{N} nj \text{ modulo } N\right] \tag{C3}$$

$$\mathbf{A} = \left[A(0)\ A(1)\ ...\ A(N-1)\right]T \tag{C4}$$

$$\mathbf{X} = \left[X(0)\ X(1)\ ...\ X(N-1)\right]T \tag{C5}$$

where the superscript T denotes a transpose of rows and columns. A row-permuted version of $\mathbf{W}_N$, denoted by the matrix product

$$\mathbf{W}_N{}^1 = \mathbf{R}\mathbf{W}_N, \tag{C6}$$

can be partitioned and factored repeatedly to obtain the FFT. The rows of $\mathbf{W}_N$ are indexed in binary notation with zero origin, and then reordered as if the indices were read in reversed

†"What is the Fast Fourier Transform?," G-AE Subcommittee on Measurement Concepts, IEEE Trans. Au-15, No. 2 (1967).

fashion with the most significant bit at the right. For example, when $N = 8$, the normal sequence of indices is 0,1,2,3,4,5,6,7 and the bit-reversed sequence is 0,4,2,6,1,5,3,7. In this case **R** has the form

$$\mathbf{R} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix} \tag{C7}$$

Partitioning $\mathbf{W}_N{}^1$ into submatrices of dimension $N/2$ and factoring yields

$$\mathbf{W}_N{}^1 = \begin{bmatrix} \mathbf{W}^1_{N/2} & 0 \\ 0 & \mathbf{W}^1_{N/2} \end{bmatrix} \begin{bmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{K}_{N/2} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} \tag{C8}$$

where **I** is the identity matrix and **K** represents a family of diagonal matrices of weighting terms

$$\mathbf{K}_{N/2} = \text{diag}\left[w^0 \; w^1 \; \ldots \; w^{N/2-1} \right] \tag{C9}$$

$$\mathbf{K}_{N/4} = \text{diag}\left[w^0 \; w^2 \; w^4 \; \ldots \; w^{2(N/4-1)} \right] \tag{C10}$$

$$\vdots$$

where $w = \exp(2\pi i/N)$.

The first factor Eq. (C8) can be partitioned and expanded similarly, and when this process is repeated to its limit, an FFT algorithm results which, however, has an important drawback:

$$\begin{bmatrix} \mathbf{W}^1_{N/2} & 0 \\ 0 & \mathbf{W}^1_{N/2} \end{bmatrix} = \begin{bmatrix} \mathbf{W}^1_{N/4} & & 0 \\ & \ddots & \\ 0 & & \mathbf{W}^1_{N/4} \end{bmatrix} \begin{bmatrix} \mathbf{I} & 0 & 0 & 0 \\ 0 & \mathbf{K}_{N/4} & 0 & 0 \\ 0 & 0 & \mathbf{I} & 0 \\ 0 & 0 & 0 & \mathbf{K}_{N/4} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{I} & 0 & 0 \\ \mathbf{I} & -\mathbf{I} & 0 & 0 \\ 0 & 0 & \mathbf{I} & \mathbf{I} \\ 0 & 0 & \mathbf{I} & -\mathbf{I} \end{bmatrix}. \tag{C11}$$

In the second factor of Eq. (C11), the **I** and **K** terms alternate along the diagonal, and in the third factor the **I** and $-\mathbf{I}$ terms alternate. Computations can be simplified by grouping similar terms together with the aid of shuffling matrices **S**. The shuffles are performed in a manner similar to cutting a deck of cards into two piles and merging them, alternating either single cards, pairs, or quadruplets, etc. Define $\mathbf{S}_1, \mathbf{S}_2, \ldots$ as follows:

$$\mathbf{S}_1\mathbf{X} = \left[X(0) \; X(N/2) \; X(1) \; \dot{X}(N/2+1 \; \ldots \; X(N-1) \right]^T, \tag{C12}$$

$$\mathbf{S}_2\mathbf{X} = \left[X(0)\ X(1)\ X(N/2)\ X(N/2+1)\ \ldots X(N-1)\right]^T, \tag{C13}$$

These matrices are all orthogonal:

$$\mathbf{S}_i^T\,\mathbf{S}_i = \mathbf{I}\ . \tag{C14}$$

A complete set of shuffles is equivalent to bit-reversed permutation; that is,

$$\mathbf{S}_{N/4}\ \mathbf{S}_{N/8}\ \ldots\ \mathbf{S}_2\ \mathbf{S}_1 = \mathbf{R}. \tag{C15}$$

For example, when $N = 8$, the sequence of indices in $\mathbf{S}_1\mathbf{X}$ is 0,4,1,5,2,6,3,7, and the sequence of $\mathbf{S}_2\mathbf{S}_1\mathbf{X}$ is 0,4,2,6,1,5,3,7, which is the $\mathbf{R}\mathbf{X}$ sequence. The second and third terms of (C11) may be written in terms of $\mathbf{S}$:

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_{N/4} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{K}_{N/4} \end{bmatrix} = \mathbf{S}_{N/4} \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{K}_{N/4} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{K}_{N/4} \end{bmatrix} \mathbf{S}_{N/4}^T \tag{C16}$$

$$\begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & -\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & -\mathbf{I} \end{bmatrix} = \mathbf{S}_{N/4} \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{I} \\ \mathbf{I} & \mathbf{0} & -\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & -\mathbf{I} \end{bmatrix} \mathbf{S}_{N/4}^T\ . \tag{C17}$$

By continued expansion,

$$\mathbf{W}_N^1 = \begin{bmatrix} \mathbf{W}_{N/4}^1 & & \\ & \ddots & \\ & & \mathbf{W}_{N/4}^1 \end{bmatrix} \mathbf{S}_{N/4} \begin{bmatrix} \mathbf{I} & & & \\ & \mathbf{I} & & \\ & & \mathbf{K}_{N/4} & \\ & & & \mathbf{K}_{N/4} \end{bmatrix} \begin{bmatrix} \mathbf{I} & & \mathbf{I} & \\ & \mathbf{I} & & \mathbf{I} \\ \mathbf{I} & & -\mathbf{I} & \\ & \mathbf{I} & & -\mathbf{I} \end{bmatrix} \mathbf{S}_{N/4}^T \begin{bmatrix} \mathbf{I} & \\ & \mathbf{K}_{N/2} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} \tag{C18}$$

where zero terms have been left blank. When carried to completion,

$$\mathbf{W}_N^1 = \begin{bmatrix} \mathbf{W}_1^1 & & \\ & \ddots & \\ & & \mathbf{W}_1^1 \end{bmatrix} \mathbf{S}_{N/4}\ \mathbf{S}_{N/8}\ \ldots \mathbf{S}_1 \begin{bmatrix} \mathbf{I} & & & \\ & \mathbf{I} & & \\ & & \ddots & \\ & & & \mathbf{K}_1 \\ & & & & \mathbf{K}_1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} \mathbf{S}_1^T$$

$$\begin{bmatrix} \mathbf{I} & & & \\ & \mathbf{I} & & \\ & & \ddots & \\ & & & \mathbf{K}_2 \\ & & & & \mathbf{K}_2 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} \mathbf{S}_2^T \ldots \mathbf{S}_{N/4}^T \begin{bmatrix} \mathbf{I} & \\ & \mathbf{K}_{N/2} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} \tag{C19}$$

The first two bracketed factors in Eq. (C19) are identity matrices $\mathbf{I}$, and the product of the first $\mathbf{S}$ matrices is $\mathbf{R}$, so that

$$\mathbf{W}_N = \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} \mathbf{S}_1^T \cdots \mathbf{S}_{N/4}^T \begin{bmatrix} \mathbf{I} & \\ & \mathbf{K}_{N/2} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix}. \tag{C20}$$

Because $\mathbf{W}_N$ is by nature symmetric, it may also be written as

$$\begin{aligned} \mathbf{W}_N = {} & \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \\ & \mathbf{K}_{N/2} \end{bmatrix} \mathbf{S}_{N/4} \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & & & \\ & \mathbf{I} & & \\ & & \mathbf{K}_{N/4} & \\ & & & \mathbf{K}_{N/4} \end{bmatrix} \mathbf{S}_{N/8} \cdots \\ \cdots & \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & & \\ & \ddots & \\ & & \mathbf{K}_2 \end{bmatrix} \mathbf{S}_1 \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} \end{bmatrix}, \end{aligned} \tag{C21}$$

which is the form used in the benchmark program.

GLOSSARY

A3, A5	Adders numbers 3 and 5 in the SPAU
$A(n)$	Spectral line, fast Fourier transform output indexed on n
ACSAR	Alternate Control Store Address Register
*	Convolution of two functions
BARA	Buffer Address Register A
BARB	Buffer Address Register B
$\{\ \}$	Brace signifies whole array; e.g., $f\{X\}$ is the Fourier transform of array X
BUFA	Buffer Store on Channel A
BUFB	Buffer Store on Channel B
Butterfly	Basic Arithmetic Operation in a radix 2 FFT
CFIL	Label in Program OCT
CONJG	Macro subsection of Program XCORREL
CTRI, CTRJ, CTRK	Counters in the SPAU
DECI, DECJ, DECK	Instructions to decrement SPAU counters
DEMOD	Label in Program OCT
DFT	Discrete Fourier transform
f	Fourier transform
FFT	Fast Fourier transform
IAT	Indirect Address Table in Program WGTSUM
INCA	Increment Register A
INCB	Increment Register B

LFFT	Symbolic name of Complex FFT Program
LPF	Low-pass filter
MCU	Microprogrammed Control Unit of the SPE
MLT	Macro subsection of Program XCORREL
N	Number of points in a data array
OCT	Symbolic name of the Coherent Demodulation and Octave Filtering Program
P1,P2,P4	Product registers in the SPAU
RAR	Ready-only memory address register
REFFT	Symbolic name of the Real FFT Program
RFIL	Label in program OCT
ROM	Read only memory (coefficient store)
SPAU	Signal Processing Arithmetic Unit of the SPE
SPE	Signal Processing Element (AN/UYK-17)
$\sim$	Complex conjugation
$T_{\ell m}$	Steering delay in Program XCORREL
W_0, W_1, W_2	Delay terms in a two-pole filter
$W(0,1,\ldots,15)$	Locations in W store
WGTSUM	Symbolic name of the Complex Weighting Program
$X(j)$	Discrete data point input to FFT, indexed on j
$X(t)$	Data input to Fourier transform, a function of t
$X(w_k)$	Channel input in Program XCORREL
XCORREL	Symbolic name of the Crosscorrelation Program
$Y(w_k)$	Beam input in Program XCORREL
$\Phi(w_k)$	Crosscorrelation output in Program XCORREL